European Sixth Framework Network of Excellence FP6-2004-IST-026854-NoE

Deliverable D2.6

Virtual Laboratory Integration Report

The EMANICS Consortium

Caisse des Dépôts et Consignations, CDC, France
Institut National de Recherche en Informatique et Automatique, INRIA, France
University of Twente, UT, The Netherlands
Imperial College, IC, UK
Jacobs University Bremen, IUB, Germany
KTH Royal Institute of Technology, KTH, Sweden
Oslo University College, HIO, Norway
Universitat Politecnica de Catalunya, UPC, Spain
University of Federal Armed Forces Munich, CETIM, Germany
Poznan Supercomputing and Networking Center, PSNC, Poland
University of Zürich, UniZH, Switzerland
Ludwig-Maximilian University Munich, LMU, Germany
University College London, UCL, UK
University of Pitesti, UniP, Romania

For more information on this document or the EMANICS Project, please contact:

Dr. Olivier Festor
Technopole de Nancy-Brabois - Campus scientifique
615, rue de Jardin Botanique - B.P. 101
F-54600 Villers Les Nancy Cedex
France
Phone: +33 383 59 30 66
Fax: +33 383 41 30 79
E-mail: <olivier.festor@loria.fr>

Document Control

Title:	Virtual Laboratory Integration Report
Type:	Public
Editor(s):	Jürgen Schönwälder, Ha Manh Tran, Iyad Tumar
E-mail:	j.schoenwaelder@jacobs-university.de
Author(s):	WP2 Partners
Doc ID:	D2.6

Contents

1 Executive Summary

This sixth "Virtual Laboratory Integration Report" presents the activities of the "Virtual Laboratory and Common Testbeds" work package (WP2) in the third phase of the EMANICS project. This phase started in January 2009 with an open call for the twelve-month period covering the time from January 2009 to December 2009. The open call for this period led to the selection of the following three projects:

1. The "EmanicsLab 3.0" project, led by the University of Zurich, integrated ten partners (UniZH, INRIA, UT, UPC, IUB, UniBwM, LMU, UCL, PSNC, UPI) and aimed at taking the necessary actions to keep EmanicsLab running during the last year of the EMANICS project and beyond the project lifetime. To this end, a federation with OneLab2, the European branch of PlanetLab, has been defined. In addition, an EmanicsLab charter has been written and agreed upon by all EmanicsLab partners. The EmanicsLab charter defines the necessary rules for a long-term operation and usage of EmanicsLab after the EMANICS project ends.

2. The "Network Trace Collection and Labeling" project, led by the University of Twente, integrated five partners (UT, INRIA, IUB, UniZH, PSNC) had multiple goals. The first goal was to extend the EmanicsLab infrastructure such that traces and be shared effectively between EmanicsLab users. Alternative approaches such as the PerfSONAR system developed by GN2 and GN3 EU-funded projects, have been investigated as well. The second goal was to provide a labelled network trace that can be used, among other things, for comparative studies of flow-based intrusion detection systems. Finally, work has been done to check the consistency of network packet traces.

3. The "Trace Visualization and Anonymization Tools" project, led by PSNC, integrated two partners (IUB and PSNC). The project focused on the evaluating of tools for trace data visualization and anonymization / deanonymization. Several visualization tools have been investigated and some new ones have been developed and a study on attacks on anonymization mechanisms have been provided.

This document describes the results achieved in these three projects.

2 Introduction

The third phase of the EMANICS project did last for twelve months, from January 2009 to December 2009. In this phase, the work package has had three objectives: the establishment of collaboration environments, the development of supporting tools, and the creation and maintenance of trace repositories for research and educational purposes. This report (deliverable D2.6) documents activities and achievements in three supported projects selected by an open call:

1. The "EmanicsLab 3.0" project, led by the University of Zurich, integrated ten partners (UniZH, INRIA, UT, UPC, IUB, UniBwM, LMU, UCL, PSNC, UPI) and aimed at taking the necessary actions to keep EmanicsLab running during the last year of the EMANICS project and beyond the project lifetime. To this end, a federation with OneLab2, the European branch of PlanetLab, has been defined. In addition, an EmanicsLab charter has been written and agreed upon by all EmanicsLab partners. The EmanicsLab charter defines the necessary rules for a long-term operation and usage of EmanicsLab after the EMANICS project ends.

2. The "Network Trace Collection and Labeling" project, led by the University of Twente, integrated five partners (UT, INRIA, IUB, UniZH, PSNC) had multiple goals. The first goal was to extend the EmanicsLab infrastructure such that traces and be shared effectively between EmanicsLab users. Alternative approaches such as the PerfSONAR system developed by GN2 and GN3 EU-funded projects, have been investigated as well. The second goal was to provide a labelled network trace that can be used, among other things, for comparative studies of flow-based intrusion detection systems. Finally, work has been done to check the consistency of network packet traces.

3. The "Trace Visualization and Anonymization Tools" project, led by PSNC, integrated two partners (IUB and PSNC). The project focused on the evaluating of tools for trace data visualization and anonymization / deanonymization. Several visualization tools have been investigated and some new ones have been developed and a study on attacks on anonymization mechanisms have been provided.

This document describes the results achieved in these three projects and is structured as follows. Section 3 documents the evolution of the EmanicsLab to what is now called EmanicsLab 3.0. Section 4 reports the traffic trace collection and labeling work performed in the project. Section 5 describes the evaluation of existing trace visualization and anonymization tools. The deliverable concludes in Section 6.

3 EmanicsLab 3.0

The main goal of the EmanicsLab 3.0 activity – besides a continuation of EmanicsLab in the last year of EMANICS – was to take the necessary actions to keep EmanicsLab running beyond the project lifetime. To this end, EmanicsLab has entered a federation with OneLab2, the European branch of PlanetLab. In addition, an EmanicsLab charter has been written and agreed upon by all EmancisLab partners. The charter defines the necessary rules for a long-term operation and usage of EmanicsLab after the EMANICS project ends.

EmanicsLab 3.0 continued with the same partners as in the previous phase. No new partners or nodes have been added to the testbed in this phase. The partners continued to operate their EmanicsLab nodes — providing maintenance and support as needed. An overview on the different sites, nodes, as well as key users and their roles can be found in the previous deliverable D2.5 and is still up-to-date. An updated overview on the usage of EmanicsLab is given in this report.

3.1 Charter

EmanicsLab is an instantiation of PlanetLab that is restricted in terms of the number of participating institutions and can therefore provide services that require resources or trust relationships that are not present in other larger scale PlanetLab installations.

EmanicsLab was initially funded and organized by the European EMANICS Network of Excellence. This document defines the rules for the future evolution of EmanicsLab that govern the further development of EmanicsLab past the end of EMANICS project. Changes to this charter have to be approved by a two-thirds majority of EmanicsLab partners.

3.1.1 EmanicsLab Administrator

The EmanicsLab Administrator is the trusted institution running myPLC for EmanicsLab and serves as the primary contact for any technical and administrative questions. The EmanicsLab Administrator is also responsible for the execution of peerings with other organizations such as PlanetLab Europe.

3.1.2 EmanicsLab Steering Committee

The EmanicsLab Steering Committee is the main responsible organization and discusses any policy related issues. It can decide on the following tasks:

- The addition and removal of EmanicsLab partner sites.
- The collaboration with other institutions such as PlanetLab Europe.
- The definition of usage and peering policies.

- The resolution of conflicts that can not be solved by the EmanicsLab Administrator.
- The approval / revocation of the EmanicsLab Administrator.

The EmanicsLab Steering Committee consists of 5 people. Steering committee members are elected by a majority vote of all EmanicsLab partners. The term is limited to three years. It is suggested that steering committee members do not rotate all at the same time.

The EmanicsLab Steering Committee elects by a majority vote the EmanicsLab Steering Committee Chair. The chair is responsible for organizing and running EmanicsLab Steering Committee meetings and he serves as the primary contact point for the EmanicsLab Steering Committee.

3.1.3 Acceptable Use Policy

The usage of the EmanicsLab infrastructure is regulated by an acceptable use policy. At the time of this writing, EmanicsLab members have to sign the PlanetLab Europe Acceptable Use Policy or the PlanetLab Acceptable Use Policy.

3.1.4 Initial Leadership Composition

The University of Zürich (Burkhard Stiller) is acting as the initial EmanicsLab Administrator. The initial EmanicsLab Steering Committee consists of top EmanicsLab users:

- Jacobs University Bremen (Jürgen Schönwälder)
- University of Catalunya (Joan Serrat)
- University of Twente (Aiko Pras)
- University of Zürich (Burkhard Stiller)
- INRIA Nancy (Olivier Festor)

The initial Steering Committee Chair is Jürgen Schönwälder (Jacobs University Bremen).

3.2 EmanicsLab Slices

In the reporting period, 19 activities have used EmanicsLab for education and research (cf. Table 1).

For each of those activities a slice has been created on EmanicsLab. The purpose of each slice is described in the following.

Slice Name	Leading Researcher
uzh_fastss	Thomas Bocek
uzh_fuse	Dalibor Peric
uzh_datta3_2	Cristian Morariu
uzh_smoothit	Fabio Hecht
uzh_coopsc	Andrei Vancea
uzh_flowshare	Andrei Vancea
uzh_sturm	Christoph Sturm
ut_snid_ut	Ramin Sadre
upc_sblomars	Josep Tomas
iub_discaria	Ha Manh Tran
iub_kadviz	Jürgen Schönwälder
inria_mape	Thibault Cholez
inria_p2psybilattack	Thibault Cholez
unibw_asam	Frank Eyermann
lmu_testdrive	Ralf König
psnc_flow3	Krzysztof Nowak
pl_elabmoni	Andrei Vancea
pl_ganglia	David Hausheer
pl_netflow	David Hausheer

Table 1: EmanicsLab Slices

3.2.1 Discaria (iub_discaria)

The iub_discaria slice is used to test the distributed case-based reasoning system, namely DisCaria. This support system aims to assist network operators in resolving faults in communication systems. DisCaria is characterized by the capability of exploring fault knowledge resources in distributed environments using peer-to-peer technology and the capability of exploiting fault solving knowledge from the resources using case-based reasoning methodology. DisCaRia peers have been installed on 11 nodes of the slide; each node contains a case-based reasoning engine for processing queries, a peer-to-peer engine for communicating between peers and a database engine for maintaining cases. About one million cases has been crawled from bug tracking systems such as Mozilla, Gentoo, Redhat, or Eclipse for the testing purpose. DisCaRia also provides web-based clients for users to submit queries (`http://emanicslab2.informatik.unibw-muenchen.de:8080/` and `http://discaria.eecs.jacobs-university.de`).

3.2.2 Kadviz (iub_kadviz)

This slice was used to collect data about the structure of the Kademlia P2P overlay. A modified Kademlia client was used to obtain topological information using an active probing technique. The goal was to feed the collected information into visualization systems. This slice was used for a student project since we needed data collectors attached to different locations in the IP routing topology.

3.2.3 MAPE (inria_mape)

This slice is used in the context of the French ANR Research Project MAPE (Measurement and Analysis of Peer-to-peer Exchanges for pedocriminality fighting and traffic profiling). We developed a distributed architecture which can monitor and control content access in the KAD DHT. We can accurately investigate the behaviour of peers interested in a specific content by monitoring, eclipsing or polluting a few number of related DHT entries on this P2P network. Our monitoring architecture is presented in the research report entitled: "HAMACK: a Honeynet Architecture against MAlicious Contents in KAD". Further results are upcoming (`http://hal.inria.fr/inria-00406477/fr/`).

3.2.4 Sybil Attack (inria_p2psybilattack)

The purpose of this slice was to execute modified aMule clients to evaluate if the KAD DHT could be cheated by new attack strategies, despite the introduction of protection mechanisms. In fact, recent versions of KAD clients (since eMule 0.49x and aMule 2.2.x) implement new preventive rules to mitigate the Sybil attack. Even if Sybil attacks from a single source are mitigated, we showed that it is still possible to take control over DHT entries with small distributed and localized attacks, as long as the KADID can be freely chosen and the Sybils placed very close to the target in the address space. This slice contributed to the experiments described in the paper entitled "Evaluation of Sybil Attacks Protection Schemes in KAD". New experiments are planned to evaluate our novel solution to mitigate such DHT attacks (`http://hal.inria.fr/inria-00405381/fr/`).

3.2.5 RN-Labs Testdrive (lmu_testdrive)

This slice has been created to investigate the use of EmanicsLab for the support of a practical lab course in computer networks at LMU. The use of EmanicsLab nodes was intended to get a better view on distributed routing, BGP routes from different perspectives in the network, and measuring round-trip delays between distributed sites. However, given the limited progress that students made in this practical course in other topics, the lab course schedule had to be changed and the slice was put on hold.

3.2.6 WP2 Flow Collect / WP7 FMAD (psnc_flow3)

This slice was used to collect NetFlow v5 data from PSNC's networks as a part of a coordinated trace collection activity and to gather data for further NetFlow-related research performed as a part of the WP7 FMAD activity. Furthermore, EmanicsLab was used in the process of PerfSONAR NetFlow tools evaluation – as declared in the WP2 Network Trace Collection and Labelling activity. The slice was used as a testbed as well as a data collection point.

3.2.7 ASAM (unibw_asam)

UniBw conducted further measurement series as part of the ASAM project in EmanicsLab. The series focused on assessing the effects of different sampling intervals and assessing the maximum sampling frequency. The most important and most voluminous measurement run sampled delay in EmanicsLab with a frequency of approx. every 2 ms (the minimum interval possible in Emanicslab with the given hard- and software).

3.2.8 SBLOMARS (upc_sblomars)

SBLOMARS is a pure decentralized monitoring system in charge of permanently capturing computational resource performance based on autonomous distributed agents. SBLOMARS was developed in a PhD thesis at UPC. It has been released under the GPL license and can be downloaded from `http://nmg.upc.edu/sblomars`. Emanicslab has been used in order to run the SBLOMARS source code in multiple nodes simultaneously.

3.2.9 WP7 SNID/FMAD (ut_snid_ut)

The SNID slice has been created to support research activities in the area of large scale network analysis and intrusion detection. For that purpose, NetFlow traffic traces from different sources have been collected and stored in relational databases running in the slice. The data has been used to develop models of malicious traffic and has served to evaluate the performance of data analysis algorithms. In addition, extensive MySQL queries were performed on flow data sets in order to find out whether there was any set of flow parameters (e.g., flow duration, flow throughput, amongst others) that could be considered as relevant estimators to predict flow size. Since these analyses required a considerable amount of computational power and memory space, EmanicsLab turned out to be a beneficial tool.

3.2.10 CoopSC (uzh_coopsc)

Semantic caching is a technique used for optimizing the evaluation of database queries by caching results of old queries and using them when answering new queries. CoopSC is a cooperative database caching architecture, which extends the classic semantic caching approach by allowing clients to share their local caches in a cooperative matter. Cache entries of all clients are indexed in a distributed data structure constructed on top of a Peer-to-Peer (P2P) overlay network. This distributed index is used for determining those cache entries that can be used for answering a specific query. Thus, this approach decreases the response time of database queries, because the server only answers those parts of queries that are not available in the cooperative cache.

3.2.11 WP9 DATTA 3 (uzh_datta3_2)

During the DATTA2 project lifetime, several shortcomings of the hierarchical approach for a distributed flow records storage platform have been identified. In order to address these

issues, a “flatter” P2P organization approach for the storage node is investigated during the third phase of the project with the following sub-activities: (1) implementation of a very light kademlia-like P2P overlay framework, (2) implementation of Distributed Flow Storage on top of the new P2P network, (3) investigation on how the new distributed platform could be embedded in network components (such as Cisco AXP cards), (4) prototypical implementation of the P2P Flow Storage platform ontop of AXP.

3.2.12 FastSS/PSH/PeerVote (uzh_fastss)

A new set of tests for the slice uzh_fastss have been carried out to evaluate an enhanced version of PeerVote with the new TomP2Pv3 framework. This time all 18 machines in EmanicsLab have been used and around 2000 peers could be deployed in total. The tests were repeated 30 times and took around 1.5 weeks to complete. This resulted in several gigabytes of data, which was analyzed. The results from these tests showed the same behavior as with previous tests with PeerVote, but this time the new PeerVote mechanism produced less overhead. A detailed report and further results are documented in Thomas Bocek’s PhD thesis.

3.2.13 FlowShare/WP2 Flow Collect (uzh_flowshare)

In order to provide trace sharing capabilities in the EmanicsLab infrastructure, UniZH has developed a generic application (dirshare) that allows a slice to share local directories with other slices. For the trace sharing scenario, a provider slice stores the trace records in a directory which will be shared with other consumer slices. The consumers’ slices will be able to mount that directory in their own file systems and read its content. Dirshare is program used for sharing directories between slices using the vsys tools. VSys allows scripts, running on the host machine, to be executed in a controlled manner from the virtual machines.

3.2.14 FUSE/WP9 P2PCOST2 (uzh_fuse)

During the previous phase, P2PCOST built a reliable infrastructure for file storage in a distributed environment. P2PCOST2 extends the current implementation by a voting mechanism to address malicious peer behavior. Furthermore, the replication and garbage collection functionalities will be moved down to the P2P layer. Additionally, this activity explores the interaction between P2PCOST and its underlying network using game theoretical models, in order to explore the possibilities for joint optimization and incentive alignment that might arise. Finally, P2PCOST2 implements globally optimal goals while maintaining a locally defined incentives structure.

3.2.15 SmoothIT (uzh_smoothit)

The University of Zurich made use of the slice uzh_smoothit to perform experiments with peer-to-peer video streaming. The main goal was to test incentive mechanisms that promote locality-awareness and improve quality of experience for the end user. The use of

EmanicsLab in this case is very important, since it offers a distributed computing environment with enough free resources to run such a resource-demanding type of application. The work performed is related to the P2PCOST2 joint activity involving UZH and UCL.

3.2.16 Peer Data Management (uzh_sturm)

The overall context of this dissertation project (`http://www.researchportal.ch/unizh/p6852.htm`) is to establish an access control mechanism in Peer Data Management Systems. The approach is to coordinate the access control components of the participants in a decentralized manner. To do this, coordination information has to be stored in the P2P network. How to do this effectively is evaluated in simulations. The simulations that will run on EmanicsLab evaluate and measure (a) the performance (reachability, overhead, number of messages, amount of transfered data, runtime) of a extended Chord based DHT when a certain percentage of the peers are malicious and collaborating, (b) the performance of a unstructured P2P Network under the same circumstances (malicious collaborating peers) using active replication with a defined replication group. The simulations should show how expensive it is to store and retrieve data in/from a P2P network in a reliable and secure way.

3.2.17 Elabmoni/WP7 VirtMon (pl_elabmoni)

Elabmoni is a monitoring application for the EmanicsLab testbed. The tool has been modified in order to be able to run in a regular EmanicsLab slice. A more detailed description of the new Elabmoni tool is provided in the next section.

3.2.18 EmanicsLab Ganglia (pl_ganglia)

The purpose and installation of Ganglia in EmanicsLab has been presented in detail in deliverable D2.4. Figure 1 is an updated Ganglia graph showing the resource usage in EmanicsLab in the reporting period. Additional figures can be found on the Ganglia website [1]. The increase in the number of CPUs is due to the fact that multi-core CPUs are counted individually by the new version of Ganglia.

3.2.19 EmanicsLab NetFlow (pl_netflow)

This slice is used to monitor the network flows on each EmanicsLab node using PlanetFlow, which has already been documented in deliverable D2.4. The new MyPLC version includes PlanetFlow2, a redesign of the former PlanetFlow tool. More information about the new version can be found at `http://www.cs.princeton.edu/~sapanb/planetflow2/`.

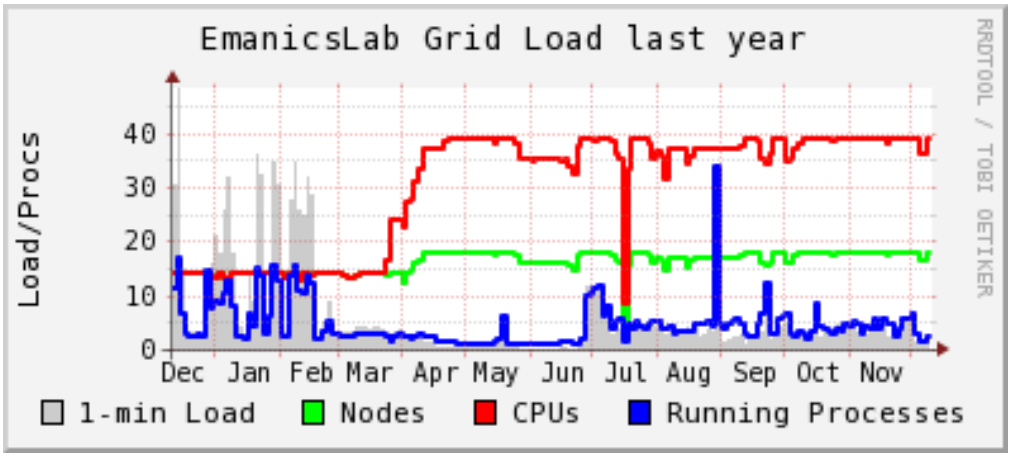


Figure 1: Resource Usage in EmanicsLab

3.3 Elabmoni

Elabmoni is one of the monitoring applications used for EmanicsLab. Its web-based interface displays information about the resource utilization of the slices running in the EmanicsLab testbed. This information is also stored in a MySQL database. The original version of the application was running in the host environment. In order to improve security, Elabmoni was modified to be able to run in a regular EmanicsLab slice. The VSys tool is used to allow the Elabmoni application to gather information about the resource utilization of other slices running on the same node. VSys allows scripts, running on the host machine, to be executed in a controlled manner from the virtual machines.

3.3.1 Architecture

Figure 2 illustrates the Elabmoni architecture. The clients are running on regular slices in different EmanicsLab nodes. As mentioned, they use the VSys tool to execute a script that gathers information about the resource utilization of slices running in the same nodes. This information is sent to the *Elabmoni collector*, which stores it in a MySQL database.

The web-based GUI was developed using PHP. It accesses the MySQL database and displays the statistical information related to the resource utilization of different slices and nodes. Figure 3 shows the Elabmoni GUI.

3.3.2 Deployment

The following steps must be executed in order to deploy Elabmoni:

- The collector application must be installed on a slice running in an EmanicsLab node. This slice should also contain the MySQL database. Figure 4 presents the schema of the table that must be created. The collector application must be configured in order to access the database.

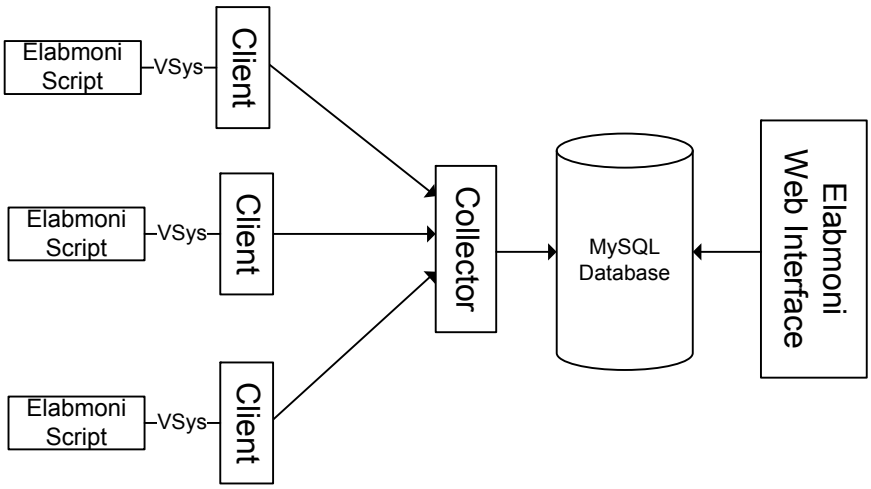


Figure 2: Elabmoni Architecture

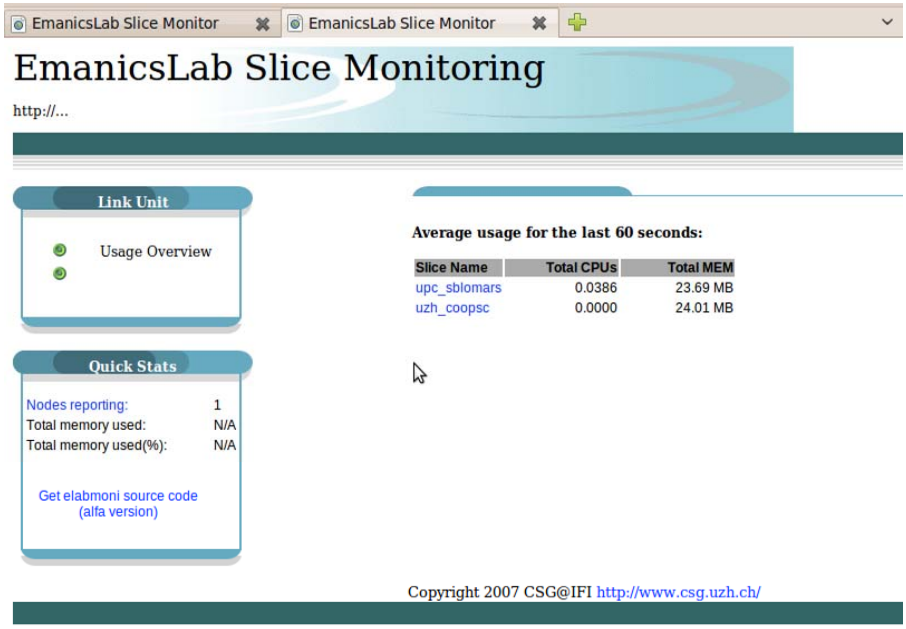


Figure 3: Elabmoni GUI

```
CREATE TABLE ‘slice-usage‘ (
 ‘timestamp‘ int(11) NOT NULL,
 ‘hostname‘ varchar(255) NOT NULL,
 ‘slicename‘ varchar(255) NOT NULL,
 ‘context‘ int(11) NOT NULL,
 ‘cpu‘ float NOT NULL,
 ‘mem‘ float NOT NULL,
 ‘rss‘ int(11) NOT NULL,
 ‘vsz‘ int(11) NOT NULL,
 KEY ‘slicename‘ (‘slicename‘),
 KEY ‘hostname‘ (‘hostname‘),
 KEY ‘timestamp‘ (‘timestamp‘)
)
```

Figure 4: MySQL Schema

- A client slice must be created in every node that is going to be monitored. The *elabmoni* VSys script must be added in the host environment. The client application should be executed with *root* credentials.
- The web-based GUI must be installed in an Apache-PHP environment. The application must be configured to access the database running on the collector slice.

3.4 Federation and Further Extensions of EmanicsLab

After INRIA established the contact with OneLab2, UniZH took the lead to discuss the federation between EmanicsLab and PlanetLab Europe (PLE) in several meetings. It was agreed that initially, EmanicsLab should federate with PlanetLab Europe only. This is to keep the load on EmanicsLab in a reasonable range (PLE at the moment includes around 50 sites). In the future, the federation may be extended to PlanetLab Central (PLC) as well. According to the current federation policy, each EmanicsLab site is able to create 10 slices on PLE sites and vice versa, like for the federation between PLE and PLC. Another meeting was held with Panayotis Antoniadis (OneLab2, LIP6) in Zurich, to discuss more complex federation policies. Such a policy could be based on the reciprocity of nodes x CPU x time. For example, for giving 10 nodes x 40% guaranteed CPU x 1 year, EmanicsLab could get 200 nodes x 10% guaranteed CPU x 3 months.

Technically, the federation has been implemented using the Slice Federation Architecture (SFA) tool also called geniwrapper. To do so, the currently installed version of myPLC in EmanicsLab was updated to v4.3. The necessary technical changes have been done in collaboration between PLE and UniZH. At the time of writing a trial of the federation has been successfully setup and tested and the real federation will be activated and made accessible to the users before the end of the project.

Legally, PL Europe and EmanicsLab Central (UniZH) wrote a memorandum of understanding (MoU) to formalize the federation. The PLE Steering Committee has already

approved the MoU with great enthusiasm. At the time of writing, the MoU is still subject to the approval by the EmanicsLab Steering Committee. Assuming it will be approved, this will complete the administrative part of the federation.

One of the requirements for federation is that the federating partners have signed a PlanetLab membership agreement (either from PLE or PLC), so they are bound to an Acceptable Use Policy (AUP). This is now the case for all EmanicsLab partners except LMU, but they are currently in the process of signing the PLE agreement as well.

To this end, documentation about the federation, the AUP, the respective incident response procedures, and the guides of PlanetLab Europe have been added to the EmanicsLab website. Additionally, a Trouble Tickets page has been added to the website with a list of current problems on the testbed, so users can go there to see if EmanicsLab admins are already aware of the problem.

3.4.1 Memorandum of Understanding

The current version of the Memorandum of Understanding (MoU) between the EmanicsLab Steering Committee and the PlanetLab Europe Consortium regarding the nature of the federation between their respective testbeds EmanicsLab and PlanetLab Europe is given below:

It is the intent of the EmanicsLab Steering Committee and the PlanetLab Europe Consortium to establish a federation between EmanicsLab, a private PlanetLab originating from the FP6 EMANICS project, run by the University of Zurich (UZH), and PlanetLab Europe (PLE), the European branch of the global PlanetLab system, run by Université Pierre et Marie Curie (UPMC).

We aim to: create the possibility for EmanicsLab users to obtain slices across EmanicsLab that also include resources from the global PlanetLab system; create the possibility for some or all PlanetLab users to obtain slices across PlanetLab that include EmanicsLab resources; increase utilisation of the EmanicsLab testbed by bringing in new users from the global PlanetLab system; possibly relieve EmanicsLab of administrative tasks associated with maintaining testbed memberships. In a first stage, the federation shall be limited to EmanicsLab and PlanetLab Europe, but can later be extended to the global PlanetLab system as soon as mutually satisfactory policies have been defined and implemented to govern the allocation of resources among the users of the different testbeds.

In doing so, we will: preserve adequate resources on EmanicsLab for EmanicsLab users to run their experiments; maintain the security of EmanicsLab data; work within the legal framework established by PlanetLab Europe and PlanetLab Central (PLC) membership agreements; maintain the principle of reciprocity under which the ability to create slices on the global PlanetLab system is provided in exchange for the contribution of resources to that system.

UZH and all other EmanicsLab sites that participate in the federation will be members of either PLE, PlanetLab Central (PLC) or an eventual EmanicsLab consortium with a similar membership agreement, and as a result will be bound by a PlanetLab acceptable use policy. Appendix 1 lists current EmanicsLab sites and those sites' membership status. Federation can start immediately with those EmanicsLab sites that are already members

of PLE or PLC, and will be extended to other sites as they fulfil the membership requirements stated above.

EmanicsLab will run an independent operations team with a "support" mailing list, and will handle local support requests, including security incidents, without the need to consult their PlanetLab Europe colleagues. For support requests that cross federation boundaries, the initial team will respond according to an agreed upon procedure/template, CC'ing the other support team to ensure that any necessary follow up is taken. The two teams will work to define a common response procedure and template messages.

Each component, such as a node, that is contributed to the global PlanetLab system, is under a management authority that determines how that component's resources are allocated to users. Part of this management authority is exercised by the site that owns the node and part of it is delegated to a central agency: UZH, Department of Informatics (IFI), for EmanicsLab and UPMC for PlanetLab Europe. EmanicsLab will regulate load on EmanicsLab nodes by reserving resources for EmanicsLab slices and by limiting the deployment of slices from the global PlanetLab system. To the extent that EmanicsLab does contribute resources to the global PlanetLab system, UPMC and other PlanetLab authorities will provide access to global PlanetLab resources that is proportional to this contribution.

Policies for resource contribution and consumption, as well as technical mechanisms to implement federation and enable the expression of those policies, are subjects for study. We will work together to develop improved policies and mechanisms.

This memorandum of understanding is a public document that may be freely copied and adapted for use by others without requesting prior permission.

Appendix 1: EmanicsLab sites

EmanicsLab sites that are members of PlanetLab:

- INRIA: Institut National de Recherche en Informatique et Automatique (PLE member)
- JUB: Jacobs University Bremen (PLC member)
- UniBw: University of Federal Armed Forces Munich (PLC member)
- PSNC: Poznan Supercomputing and Networking Center (PLC member)
- UPC: Universidat Politechnica de Catalunya (PLE member)
- UCL: University College London (PLC member)
- UZH: University of Zurich (PLC member)
- UPI: University of Pitesti (PLE member)
- UT: University of Twente (PLE member)
- LMU: Ludwig-Maximilian University Munich (PLE member[1])

[1] At the time of writing, PLE membership of LMU is subject to final signature

4 Network Trace Collection and Labeling

The availability of network traces is crucial for the research of communication systems. Traces are necessary to understand current communication behaviors, for evaluation purposes, as input for simulation, etc. An important aspect is the sharing of traces among researchers. In general, several questions and problems arise when sharing traces. In the following, several of them are addressed.

Section 4.1 describes how trace-sharing capabilities can be provided in the EmanicsLab infrastructure. A generic application has been developed that allows a slice to share local directories with other slices.

In Section 4.2, a flow data set is presented in which security-related anomalies have been marked. Such so-called *labeled* data sets are important for tuning and comparative evaluation of intrusion detection systems. As far as we know, this labeled flow data set is the first publicly available one and it has found some interest in the research community since its presentation in [2].

Using publicly available data can be very convenient for a researcher, as gathering data yourself can be very time-consuming, or even impossible. A potential issue in using a repository is the consistency of the traffic inside the repository. In Section 4.3, an approach to detect such inconsistencies is proposed.

In Section 4.4, we evaluate the perfSONAR NetFlow tools and assess its applicability to the EMANICS trace collection activities as a tool to easily subscribe and request different NetFlow sources and store them and/or present to the user.

4.1 Sharing Traces in EmanicsLab (UniZH)

In order to provide trace sharing capabilities in the EmanicsLab infrastructure, UniZH has developed a generic application (*dirshare*) that allows a slice to share local directories with other slices. For the trace sharing scenario, a *provider* slice stores the trace records in a directory which will be shared with other *consumer* slices. The consumers slices will be able to mount that directory in their own file systems and read its content.

Dirshare is a program used for sharing directories between slices using the *vsys* tools[2]. VSys allows scripts, running on the host machine, to be executed in a controlled manner from the virtual machines. Each virtual machine that was configured to use dirshare has two name pipes (”dirshare.in” and ”dirshare.out”) in the /vsys directory. The file ”dirshare.in” is used for sending data to the dirshare script (which is running on the host machine) while ”dirshare.out” is used for getting the result of the execution.

4.1.1 Sharing a Directory

In order to share a directory from a virtual machine, the following command must be executed:

[2] `http://www.cs.princeton.edu/~sapanb/vsys/`

```
echo "share sharename dirname slice1,slice2,..,sliceN" \
     > /vsys/dirshare.in
```

Arguments:

- *sharename* - the name with which other slices will refer to the shared directory
- *dirshare* - the name of the directory which is shared
- *slice1,slice2,..,sliceN* - the slices that can mount the shared directory

Example:

```
cat /vsys/dirshare.out &  #for displaying the result on the console
echo "share flows /home/collector/flows uzh_coops,uzh_test2" \
   > /vsys/dirshare.in
```

4.1.2 Unsharing a Directory

The following command unshares a directory:

```
echo "unshare sharename" > /vsys/dirshare.in
```

Arguments:

- *sharename* - the name of shared resource

Example:

```
cat /vsys/dirshare.out &
echo "unshare flows" > /vsys/dirshare.in
```

4.1.3 Mounting a Directory

The following command must be executed for mounting a shared directory. It will only be successful if the current virtual machine is allowed to mount the shared directory.

```
echo "mount sourceslice sharename localdir" > /vsys/dirshare.in
```

Arguments:

- *sourceslice* - the name of the slice that shared the directory
- *sharename* - the name of the shared resource
- *localdir* - the local directory where the shared directory is mounted (it must exist)

Example:

```
cat /vsys/dirshare.out &
echo "mount uzh_flowstore flows /home/app/flows" > /vsys/dirshare.in
```

4.1.4 Unmounting a Directory

The following command unmounts a shared directory:

```
echo "umount localdir" > /vsys/dirshare.in
```

Arguments:

- *localdir* - the local directory where the shared resource is mounted

Example:

```
cat /vsys/dirshare.out &
echo "umount /home/app/flows" > /vsys/dirshare.in
```

4.2 A Labeled Network Trace

In 2008, activities have been initiated (see deliverables D2.5) with the goal to create labeled flow data sets that can be used, among others, for comparative studies of flow-based intrusion detection systems. The activity has been completed in 2009 and its results have been published in [2] and are reported in the following.

4.2.1 Introduction

Considering the increasing number of security incidents and discovered vulnerabilities per year [3], it is not surprising that intrusion detection (ID) has become an important research area in the last decade. A large number of ID techniques have been proposed and many of them have been implemented as prototypes or in commercial products. Moreover, the research community has recently focused on flow-based approaches.

When proposing a new intrusion detection system (IDS), researchers usually evaluate it by testing it on labeled (or annotated) traffic traces, i.e., traffic traces with known and marked anomalies and incidents [4]. Labeled traces are important to compare the performance of diverse detection methods, to measure parameter effectiveness and to fine-tune the systems. Ideally, a labeled traffic trace should have the following properties: it should be realistic (opposed to "artificial"), completely labeled, containing the attack types of interest and, not less importantly, publicly available. Despite the importance of labeled traces, research on IDS generally suffers of a lack of shared data sets for benchmarking and evaluation. Moreover, we have no knowledge of any publicly available *flow-based* traffic trace that satisfies all these criteria.

Several difficulties prevent the research community to create and publish such traces, in first place the problem of balancing between privacy and realism. It is natural that the most realistic traces are those collected "in the wild", for example at Internet service providers or in corporate networks. Unfortunately, these traces would reveal privacy sensitive information about the involved entities and hence are rarely published. On the other hand, artificial traces, i.e., traces that have not been collected but artificially generated, can avoid the problem of privacy but they usually require higher effort and deeper domain knowledge to achieve a realistic result. Moreover, labeling is a time consuming process: it could easily be achieved on short traces, but these traces could present only a limited amount of security events. Therefore, most publications use non-public traffic traces for evaluation purposes. The only notable exception are the well-known DARPA traces [5, 6, 7], which still are, despite their age, the only publicly available labeled data sets specifically created for intrusion detection systems evaluation.

In this section, we present the first labeled flow-based data set. We describe how a suitable traffic trace can be collected and how it can be labeled. A flow is defined as "a set of IP packets passing an observation point in the network during a certain time interval and having a set of common properties" [8]. Our final data set will contain only flows and full-packet content will be used only as additional information source during the labeling process. Concentrating on flow-based data sets is in our opinion important for mainly two reasons. First, it substantially reduces the privacy concerns compared to a packet-based

data set: since there is no payload, it is possible to deal with privacy issues by anonymizing the IP addresses. Second, several promising results have been recently proposed in the flow-based intrusion detection research community (for example, [9, 10, 11]): we feel that especially now there is the need for a shared flow data set.

Publications on the labeling of non-artificial data traces are rare and, as said, we are not aware of any concerned flow-based data sets. WebClass [12], a web-based tool that allows people to contribute to manually label traffic time-series, has not found much interest in the community. More effort has been done in the creation of artificial data sets. The already mentioned DARPA trace consists of generated traffic, equivalent to the traffic at a government site containing several hundreds of users on thousands of network hosts. In [13], the authors propose the synthetic generation of flow-level traffic traces, however without discussing its technical implementation. The malicious-traffic generation in the MACE [14] and FLAME [15] frameworks are performed by mixing background traffic with the output of different attack and anomalies modules. In MACE, it is up to the user to create a realistic traffic trace by providing an appropriate configuration file.

4.2.2 Data collection

A proper measurement setup depends on the requirements we expect the labeled data set to meet. In our case, we want the data set to be *realistic*, as *complete* in labeling as possible, *correct*, achievable in a acceptable *labeling time* and of a sufficient *trace size*. In our research, we studied diverse approaches to data collection. In the following we present our operational experience, explaining strengths and drawbacks of possible measurement infrastructures according to our data set requirements. In addition, we describe the technical details of our measurement setup.

Operational experience in data collection

This subsection will discuss the impact of both *measurement scales* and *flow collection location* on our requirements.

Measurement scale Flows can be collected at different measurement scales. The University of Twente has a 10 Gbps optical Internet connection with an average load of 650 Mbps and peaks up to 1.0 Gbps. Several hundred million flows are exported per day [10]. Traces collected on this network would for sure be realistic, but we cannot accomplish the goals of completeness in labeling and reasonable creation time. Labeling network-wide traces suffers of scalability issues.

Let us now concentrate on a small subnetwork that is primarily used by us for research purposes. Due to the limited number of users, it should be easy to distinguish trusted IP addresses from unknown ones, leaving out only a small fraction of suspicious traffic to be further analyzed. However, our findings show that more than 60% of the connections cannot easily be categorized as malicious or benign. Collecting on a small subnetwork ensures us to have a realistic data set, but, as for the network scale, it would be neither complete nor achievable in a short time.

A different setup, and the one that we finally chose, is based on monitoring a single host with *enhanced logging* specifically tuned to track malicious activities, e.g., a *honeypot*. A honeypot can be defined as an “environment where vulnerabilities have been deliberately

introduced to observe attacks and intrusions" [16]. In this case, the trace size would be smaller and the labeling time limited. Moreover, an everyday service setup would ensure the traffic to be realistic. Finally, the logs would ensure us to achieve both completeness and correctness in labeling.

Flow collection location We have different options about the flow collection location. A possibility is to collect the flows generated by a NetFlow enabled router, in our case the University one. However, decoupling the flow creation from the log location introduces errors in measurements, such as timing delays and split TCP sessions. These issues make impossible to properly associate a security event with the flow that caused it. A data set based on these premises would have a serious lack in completeness and correctness. Another possibility is therefore to dump only the traffic reaching the honeypot and to create the flows off-line after the data collection is completed. This decision allows to have complete control over the flow-creation process, overcoming the problem of the session splitting.

Discussion We will now summarize the methods we studied for the data set collection, showing if they meet our requirements. Please note that, since monitoring the university network or a subnetwork does not scale, we did not explore further the options of online/offline flow creation. All our approaches ensure the trace to be realistic. Moreover, monitoring the University network or a subnetwork would also provide sufficiently large traces. After we measured the traffic reaching our honeypot, we can say that also in this case this requirement can be met. Regarding completeness and correctness, large network traces reduce the trust we can have on the labeled data set. The honeypot approach is more reliable since it offers additional logging information, but in this case, the flow collection/creation is crucial. As previously in this section, relying on external flow sources can introduce measurement errors. Creating the flows offline, on the other hand, allows to have flows that better suit the needs of a labeled data set. Finally, large infrastructures suffer of scalability in labeling time, while the honeypot setup overcomes this problem.

From this section, we can conclude that monitoring a single host that has enhanced logging capabilities is a promising setup for flow-based data sets.

Experimental setup

Our honeypot was installed on a virtual machine running on Citrix XenServer 5 [17]. The decision to run the honeypot as a virtualized host is due to the flexibility to install, configure and recover the virtual machine in case it is compromised. In addition, a compromised virtual machine can be saved for further analysis. Diverse scenarios are possible, in terms of number of virtual machines, operative systems and software. Our experimental setup consisted of a single virtual machine, on which we installed a Linux distribution (Debian Etch 4.0). In order to keep the setup simple, controllable and realistic, we decided not to rely on honeypot software, but to configure the host by ourselves. In particular, the following services have been installed:

- `ssh`: OpenSSH service [18]. Beside the traditional service logs, the OpenSSH service running on Debian has been patched in order to log sessions: for each login, the *transcript* (user typed commands) and the timing of the session have been recorded. This patch is particularly important to track active hacking activities.

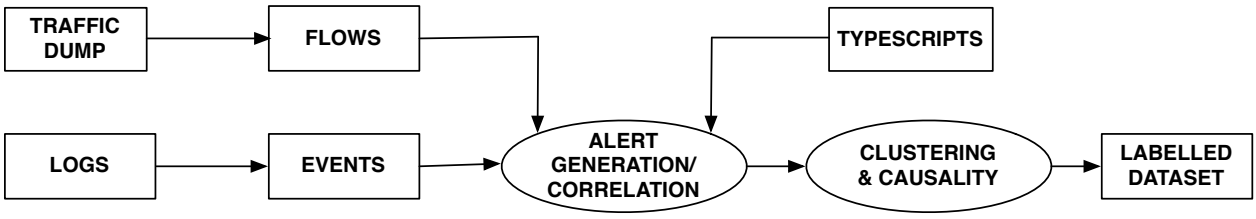


Figure 5: From raw data (packets and logs) to the labeled data-set

- Apache web server: a simple webpage with a log in form has been deployed. We relied on the service logging capabilities for checking the content of the incoming `http` connections.
- `ftp`: The chosen service software was `proftp` [19]. As for `http`, we relied on the `ftp` logs for monitoring attempted and successful connections. `proftp` uses the `auth/ident` service running on port 113 for additional authentication information about incoming connections.

Along with the service logs, we decided to dump all the traffic that reached the honeypot during our observation period. The virtual machine ran for 6 days, from Tuesday 23 September 2008 12:40:00 GMT to Monday 29 September 2008 22:40:00 GMT. The honeypot was hosted in the university network and directly connected to the Internet. The monitoring window is comprehensive of both working days and weekend days. The data collection resulted in a 24 GB dump file containing 155.2 million packets.

4.2.3 Data processing and labeling

The labeling process enriches the data trace with information about (i) the type and structure of the malicious traffic, (ii) dependencies between single isolated malicious activities. The latter is particularly important for a flow-based data set where by design no further detail on the content of the communication is available to the end user. In this section, we describe the processing of the collected traffic data and how the labeling information is generated and structured.

Figure 5 gives an overview on the whole data processing and labeling. As a first step, shown in the left part of the figure, the collected traffic dumps are converted into flows. In addition, the data extracted from the diverse log files is converted into a common format (log events) that simplifies the following processing steps. The resulting flow records and log events feed the alert generation/correlation process. Finally, a post-processing step generates additional information, namely the so-called cluster alerts and the causality information. The different steps are explained in the following.

From packets to flows

The first step is the creation of flows from the traffic trace. In our data set, a flow closely follows the Netflow v5 definition and has the following form:

$$F = (I_{src}, I_{dst}, P_{src}, P_{dst}, Pckts, Octs, T_{start}, T_{end}, Flags, Prot)$$

where the unidirectional communication is defined by the source and destination IP addresses I_{src} and I_{dst}, the employed ports P_{src} and P_{dst} (in case of UDP/TCP traffic), and the transport layer protocol type $Prot$ (treating ICMP as a transport layer protocol for simplicity). The fields $Pckts$ and $Octs$ give the total number of packets and octets, respectively, in the data exchange; the TCP header flags are stored as a binary `OR` of the flags in all the packets of the flow (field $Flags$); the start and end time of the flow are given in T_{start}, respectively, T_{end} in millisecond resolution. The flow creation has been performed using a modified version of `softflowd`[20].

From log files to log events

Information about attacks against our honeypot machine can be extracted from the log files of the services we were monitoring. In order to simplify the alert generation process, the relevant data found in the various log files is converted into log events. A log event consists of the following information:

$$L = (T, I_{src}, P_{src}, I_{dst}, P_{dst}, Descr, Auto, Succ, Corr)$$

where T gives the timestamp of the attack (as found in the logs), I_{src} and P_{src} give the IP address and used port (if available) of the attacker, and I_{dst} and P_{dst} give the attacked IP address (our honeypot) and port number. In addition, a deeper analysis of the log files reveals whether an attack was automated or manual and succeeded or failed (flags $Auto$ and $Succ$, respectively). The field $Descr$ allows us to enrich the data set with additional information retrieved from the log files. The field $Corr$ is always initialized to $false$ and later used by the alert generation process.

Alert generation and correlation

The goal of this step is to generate so-called alerts and to correlate each generated alert to one or more flows. An alert describes a security incident and is represented by the following tuple:

$$A = (T, Descr, Auto, Succ, Serv, Type)$$

The fields T, $Descr$, $Auto$, $Succ$ are defined as for the log events (see Section 4.2.3). The field $Serv$ gives the service addressed by the security incident, for example `ssh` or `http`. The $Type$ field describes the type of the incident. The alert generation process consists of three steps that are explained in the following.

Alerts from log events For attacks toward the honeypot, alerts can be directly generated from the log events. The fields T, $Descr$, $Auto$, $Succ$ of an alert record are set to the values of the corresponding fields of the log event. The $Serv$ field is set accordingly to the destination port field of the log event, for example $Serv =$ `ssh` if $P_{dst} = 22$. The $Type$ field is always set to the value `CONN`, indicating that the alert describes a malicious *conn*ection attempt.

In order to correlate an alert with a flow, we have to find the flow that corresponds to the log event from which the alert has been generated. This is not a trivial task since the timing information extracted from a log file may not be aligned with the flow's one. In addition, we would like to not only correlate the incoming (as seen from the honeypot) flow to the alert but also the response flow of the honeypot.

We use the flows as starting point for the alert generation and correlation. This avoids that a flow is correlated to more than one alert. The resulting procedure for a service s is

Algorithm 1 Correlation procedure

1: **procedure** ProcessFlowsForService (s : service)
2: **for all** Incoming flows F_1 for the service s **do**
3: Retrieve matching response Flow F_2 such as
4: $F_2.I_{src} = F_1.I_{dst} \wedge F_2.I_{dst} = F_1.I_{src} \wedge F_2.P_{src} = F_1.P_{dst} \wedge F_2.P_{dst} = F_1.P_{src} \wedge$
5: $F_1.T_{start} \leq F_2.T_{start} \leq F_1.T_{start} + \delta$
6: with smallest $F_2.T_{start} - F_1.T_{start}$;
7: Retrieve a matching log event L such as
8: $L.I_{src} = F_1.I_{src} \wedge L.I_{dst} = F_1.I_{dst} \wedge L.P_{src} = F_1.P_{dst} \wedge L.P_{dst} = F_1.P_{src} \wedge$
9: $F_1.T_{start} \leq L.T \leq F_1.T_{end} \wedge$ **not** $L.Corr$
10: with smallest $L.T - F_1.T_{start}$;
11: **if** L exists **then**
12: Create alert $A = (L.T, L.Descr, L.Auto, L.Succ, s, \texttt{CONN})$.
13: Correlate F_1 to A ;
14: **if** F_2 exists **then**
15: Correlate F_2 to A ; $L.Corr \leftarrow \mathbf{true}$;
16: **end if**
17: **end if**
18: **end for**

shown in Algorithm 1. As a first step, a best matching response flow is selected for each flow of the considered service (lines 4.2.3-4.2.3). The matching is made based on the flow attributes. If there are more than one candidate flows, the closest in (future) time is chosen. It is possible, nevertheless, that such a flow does not exist, for example in the case in which the target was a closed destination port. Since the flow tuple source/destination address/port may appear multiple times in the data set, the parameter δ ensures that an incoming flow is not correlated with a response too late in time. Values of δ from 1 to 10 seconds are possible.

After searching for a response flow, the algorithm proceeds with retrieving the best matching log event (lines 4.2.3-4.2.3). The log event must match the flow characteristics. The timing constraint in this case forces the log event to be in the interval between the beginning and the end of the flow. Moreover, since log files do not provide us with millisecond precision and multiple alerts can be generated by the same host in the same second, we require the matching log event to not have been consumed before (line 4.2.3). If the matching event does exist, the algorithm will create the alert, correlate it with the flow and, if possible, with the response flow (lines 4.2.3-4.2.3). Finally, the log event will be marked as consumed (line 4.2.3).

Alerts for outgoing attacks Some of the incoming attacks against the `ssh` were successful. As a consequence, the attacker used the honeypot machine itself to launch `ssh` scans and dictionary attacks against other machines. In order to generate alerts for these outgoing attacks, we have analyzed the typescripts of the `ssh` sessions. This allowed us to reconstruct the times and the destinations of the different attacks launched from the honeypot. Similarly to the previous step, we have used this information to find the corresponding flows and to correlate them to alerts of type `CONN`.

Alerts for side effects Several attacks towards and from the honeypot have caused non-malicious network traffic that we consider as "side effects". Most notably, `ssh` and `ftp`

connection attempts caused ICMP traffic and traffic directed to the auth/ident service (port 113) of the attacker, respectively, the honeypot. Furthermore, one attacker installed an IRC proxy on the honeypot. For these flows, we have created alerts of type SIDE_EFFECT with $Serv =$ ICMP, respectively, $Serv =$ auth or $Serv =$ IRC.

Generation of cluster alerts and causality information

The alerts described in the previous section represent single security incidents and are directly correlated with one or more flows. In addition to those basic alerts, we also generate so-called cluster alerts and extra-information describing the causal relationships between alerts.

Cluster alerts are used to label logical groups of alerts. For example, in the case of an ssh scan consisting of 100 connection attempts, we create basic alerts of type CONN for each connection, plus *one* cluster alert of type SCAN for the entire scan operation. More formally, a basic alert A belongs to a scan alert C, if (i) the alert A has the same source IP and type as the other alerts in the cluster, and (ii) the alert is not later than γ seconds after the latest alert in the cluster. In our case we set $\gamma = 5$ seconds.

As a final step, we manually add causality information. In the current setup, that means that we have created (i) links between the alert representing an attacker's successful log-in attempt into the honeypot via ssh and all alerts raised by the attacker during that ssh session, and (ii) links between the alerts of the ICMP and auth/ident flows and the alerts of the ssh and ftp flows that caused them.

Our data set has been implemented in a MySQL database. The structure of the database reflects the alert and flow structure and the relations between these categories.

4.2.4 The Labeled Data Set

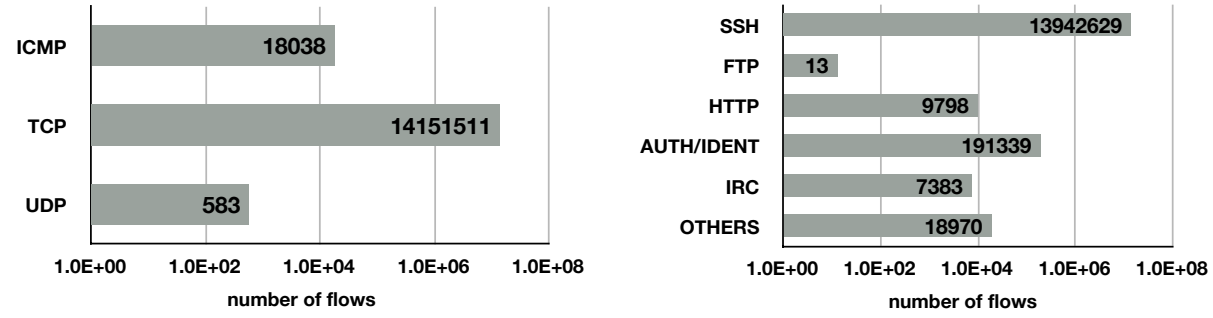
The processing of the dumped data and logs, collected over a period of 6 days, resulted in 14.2M flows and 7.6M alerts.

Flow and Alert breakdown

Figure 6(a) and 6(b) present a breakdown of the flows according to the transport layer protocols and the most active services, respectively.

At network level, the collected data set presents a subdivision in only three IP protocols: ICMP, TCP and UDP. The majority of the flows has protocol TCP (almost 99.9% of the entire traffic). A second slice of the data set consists of ICMP traffic (0.1%). Finally, only a negligible fraction is due to UDP traffic. The transport layer protocol breakdown is consistent with the services flow breakdown, shown in Figure 6(b). The honeypot most active service has been ssh, followed on the distance by auth/ident.

Figures 7(a) and 7(b) show the alert breakdown according to malicious connections and scans on the most active services. ssh and auth/ident are the only services for which the honeypot has been both a target and a source. The honeypot received several thousand ssh connections and it has been responsible of millions of outgoing ones. On the contrary, it has been the target of extensive auth/ident requests, but it produced only 6 outgoing connection to port 113. As shown in Figure 7(b), the ssh alerts can be grouped into 45



(a) (b)

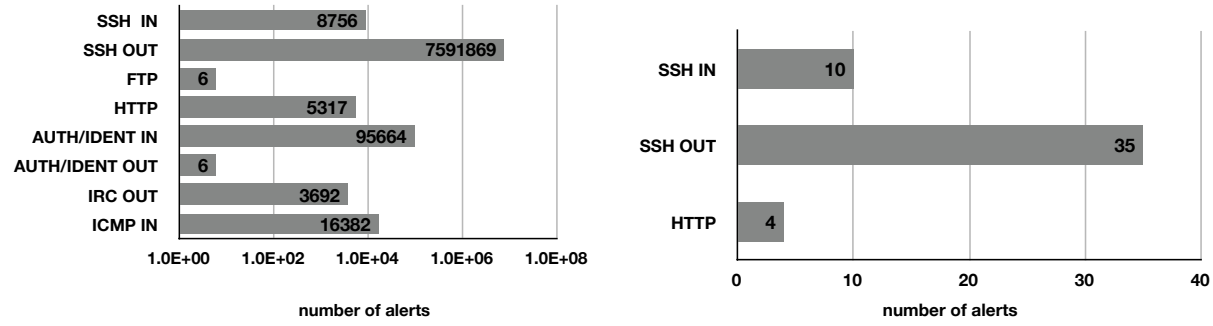
Figure 6: Flow breakdown according to the transport protocol (a) and the service traffic (b)

scans, 10 incoming and 35 targeting remote destinations. We also have been the target of 4 `http` scans. None of the `ftp`, `auth/ident` or `irc` alerts can be clustered into scans.

Honeypot target and source behaviour

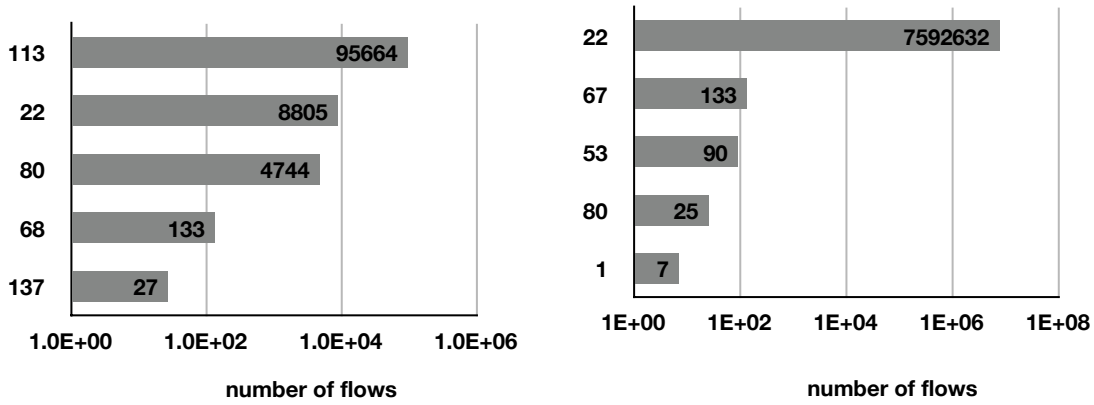
Figure 8(a) presents the five most contacted ports on the honeypot. Port 113 (`auth/ident` service) is the most targeted one. Among the services we monitored, the most often contacted ones are `ssh` (port 22) and `http` (port 80). On these ports we actually received malicious traffic, confirming our initial idea that daily used services are normally target of attacks. The incoming traffic on port 68 is due to periodical renew of the dynamically assigned IP address (`dhcp`). Finally, we received traffic on port 137 due to the `netbios` protocol, to which we never responded.

Figure 8(b) presents the top 5 ports contacted by the honeypot on remote destinations. The hit list is opened by `ssh`, confirming the alert breakdown in Figure 7. Traffic directed to port 67 has been generated by `dhcp` and it matches the incoming traffic on port 68. The traffic on port 53 (`dns`) was directed to the University `dns` servers. The outgoing traffic towards port 80 consists of only RST packets. The dumped traffic shows that remote hosts contacted us with SYN/ACK packets from port 80. Since the honeypot never initiated this interaction, it reset the connections. For its characteristics, this traffic seems not to be



(a) (b)

Figure 7: Alert repartition for basic (a) and cluster (b) types.



(a) (b)

Figure 8: Top 5 contacted ports. Figure (a) presents the incoming traffic, while Figure (b) the outgoing one (logarithmic scale).

related to attacks: we suspected it is a form of background radiation [21, 22], or part of a SYN/ACK scan. Traffic to port 1 is negligible (`tcpmux`).

Discussion on the data set

This section will discuss the results we obtained as outcome of the correlation process, pointing out the characteristic of both the labeled and unlabeled traffic. Our correlation process succeeded in labeling more that 98.5% of the flows and almost the totality of the alerts (99.99%).

Malicious traffic All the labeled traffic is related to the monitored services. Since we did not interfere in any way with the data collection process, i.e., we avoided any form of attack injection and we did not advertise our infrastructure on hacker chats, we can assume that the attacks present in the data set reflect the situation on real networks.

The majority of the attacks targeted the `ssh` service and they can be divided into two categories: the automated and the manual ones. The first ones are well-known automated brute force scans, where a program enumerates usernames and passwords from large dictionary files. This attack is particularly easy to observe at flow level, since it generates a new flow for each connection. Attacks of the second type are manual connection attempts. There are 28 of these in our trace and 20 of them succeed.

The `http` alerts labeled in our data set are automated attacks that try to compromise the service by executing a scripted series of connections. These scans are executed using tools like Nikto and Whisker, both easily available online. Differently from the `ssh` connections, no manual `http` attacks are present.

Regarding the `ftp` traffic, the data set contains only 6 connections to this service on the honeypot, during which a `ftp` session has been opened and immediately closed. Even if the connections did not have a malicious content, this behavior could be part of a reconnaissance performed by an attacker gathering information about the system.

Side-effect traffic Part of the traffic in our data set is the side effect of attacks but cannot be considered by itself malicious. `auth/ident`, ICMP and `irc` traffic fall in this category.

The scanning activities have been responsible of the majority of the flows to and from port 113. The service is supposed, indeed, to retrieve additional information about the source of a `ssh` connection. Regarding the ICMP traffic, more that 120000 incoming packets have type *time exceeded* and *destination unreachable* and come from networks that the honeypot scanned.

The analysis of the honeypot showed that a hacker installed an IRC proxy that received chat messages from several channels. It does not appear anyway that any message has been sent from our honeypot or that the channels have been used for any malicious activity. Even if this traffic is due to an application that we did not originally install on our machine, we decided not to label it as malicious but as a side effect.

Unknown traffic and uncorrelated alerts For a small fraction of the data set, we cannot establish the malicious/benign nature of the traffic. This traffic consists of three main components: (i) `ssh` connections for which it has not be possible to find a matching alert, (ii) heterogeneous traffic, containing all the flows having as destination a closed honeypot port and (iii) some not malicious connections to the `http` and `vnc` services. The `vnc` service was accessible because of the XEN virtualization software running on the honeypot. Strangely, no attacker identified this service and tried to compromise it by using known vulnerabilities or performing a brute-force dictionary attack on the password . The `vnc` flows in the data set are originating from two hosts which contacted the service but did not complete the connection.

Regarding the alerts, for few of them (0.01%) no matching traffic has been found. This makes us aware that during the collection phase some packets that reached our honeypot have not been recorded by tcpdump.

4.2.5 Conclusions

The main contribution of this work is to present the first labeled data set for flow-based intrusion detection. While approaching the problem of creating a labeled data set, we consider the choice of the proper data collection infrastructures a crucial point. It has indeed impact not only on the feasibility of the labeling, but also on the reliability of the result. We studied several data collection infrastructures, enlightening strengths and drawbacks. We conclude that, in the case of flow-based data sets, the most promising measurement setup is monitoring a single host with enhanced logging capabilities. In the specific context of this experiment, the host was a honeypot. The information collected permitted us to create a data base of flows and security events (alerts).

We have also described a semi-automated correlation process that matches flows with security events and, in addition, reflects the causality relations between the security events themselves. The results of the correlation process show that we have been able to label more that 98% of the flows in the trace. The correlation process also proves that, although we limited our measurements to a single host, labeling remains a complex task that requires human intervention.

The approach we have proposed allows us to capture unexpected security events, such as the behavior of a compromised machine. However, the presented trace mainly consists of malicious traffic. For evaluation of an IDS, this means that our data set allows to detect

false negatives but not false positives. In the future, we aim at extending our monitoring setup to more challenging scenarios. An example would be a server that is daily accessed by benign users. Monitoring such a server would result in a trace with a more balanced content of malicious and normal traffic. We believe that an approach similar to the one we presented here will be useful also in this scenario. Finally, the data set is available on e-mail request to the authors.

4.3 Consistency of Network Traffic Repositories

Traffic repositories with TCP/IP header information are very important for network analysis. Using a repository can be very convenient for a researcher, as gathering data yourself can be very time-consuming, or even impossible. A potential issue in using a repository is the consistency of the traffic inside the repository. When traffic inside a repository does not completely correspond with the actual traffic that was transmitted and received, this can influence measurements, analysis and therefore also conclusions that researchers draw. Hence, it is critical to have information about the consistency of the network traffic repository, so it can be taken into account when analysing the data.

In the following, an algorithm to detect such inconsistencies is proposed. Note that the reported results have been also published in [23].

4.3.1 Introduction

Issues with consistency of the data in repositories have been reported by Timmer in [24]. While using a repository, it appeared that not all data was recorded properly. Timmer introduces the term “fake gap” to represent those parts of a TCP flow that are absent in the repository, although they were acknowledged at the TCP level. In [25] a relatively simple algorithm has been developed to find a sudden decrease in data in small intervals, which may indicate a problem with the repository. However it may very well be a temporary network problem and therefore does not necessarily affect the consistency of the repository. Although [25] has performed some initial research, the consistency analysis never exceeded a few data files. However, as researcher, knowledge about inconsistency in a repository is essential. At this moment, there are no statistics about possible inconsistency of repositories available. In this section we will therefore analyse two well-known repositories: the WIDE and Simpleweb repositories. A tool has been developed that analyses TCP flows. We focus on TCP, because its state-full nature allows detection of fake gaps; for UDP this is, due to its stateless nature, not possible.

The main question that will be answered in the following is: *How can inconsistency be detected in a TCP traffic repository?* To answer the main question, we first focus on detecting fake gaps by introducing an algorithm. The sub question we will answer is: *How can we detect fake gaps?* Next, we built a prototype of the algorithm to test existing repositories in order to answer the second sub question: *How consistent are today’s repositories?*

The structure of this section is as follows. In Section 4.3.2 we will propose an algorithm to detect inconsistency. In Section 4.3.3 we will discuss the results of testing two existing repositories using a prototype we built and finally in Section 4.3.4 we will answer our research questions, draw conclusions and discuss possible future work.

4.3.2 Detecting Inconsistency

In this Section, we will describe inconsistency called fake gaps and introduce an algorithm for detecting inconsistency. Although there may be different ways a repository can be inconsistent, we will only consider fake gaps.

Fake Gaps

We said a fake gap to be representing those parts of a TCP flow that are absent in the repository, although they were acknowledged at the TCP level. To explain this more precisely we first have to consider a gap. To start with an example: when Alice wants to send some data to Bob, she sends ordered packets named A B C D E F. Bob may receive this as A B C E F **D**, but knows about the correct order and can therefore recreate the original message. This is called packet reordering. At the side of Bob, after having received C, there is a gap until D is received. In this example, the gap is filled when D is received.

We call a gap a fake gap, when one or more packets in a sequence of packets are not present, but are also not retransmitted; hence the original TCP flow is not affected. Consider Alice and Bob: Alice sends the sequence A B C D E F to Bob using TCP. Bob's network administrator records all traffic sent to Bob. According to the data recorded, Bob received A B C E F. When the connection between Alice and Bob is closed, we know Bob must have received D. We can then say that the recorded TCP flow between Alice and Bob contains a fake gap. All data was transmitted and received correctly, but the recorded data does not reflect this. Hence the recorded data is inconsistent. From now on, if we are referring to the flow of data packets within a TCP connection, we will abbreviate it to a flow.

In recorded traffic data, it is likely that there are flows that start and/or end outside the recorded time period. Therefore, detecting fake gaps in these flows is difficult. To avoid this, we only take flows into account that are sufficiently recorded. All packets of a single flow, from the first SYN-packet up to a FIN- or RST-packet should be recorded. If this is the case, we call the TCP flow a *usable flow*. Note that the final FIN-handshake can be partly outside the recorded time period.

Algorithm

In Section 4.3.2 we concluded fake gaps prove inconsistency. To detect this the algorithm we introduce, listed as Fig. 9, first extracts all usable flows from the complete set of packets. The second part of the algorithm loops over all usable flows. For every usable flow, it checks if there is an acknowledging packet that acknowledges a packet that has not yet been seen. If so, this indicates a fake gap and an identifier of the flow together with the packet that is used to detect the fake gap, is added to a list. So the final result of this algorithm is a list of flows combined with packets directly after the fake gaps.

It is important to see that this algorithm alone cannot detect the exact amount of fake gaps within the traffic dump. It can give an indication and show which usable flows are affected. If, for example, during a small interval no packets were recorded at the recording device, multiple flows can be affected by this. Our algorithm would detect a fake gap for each affected flow. We do not consider this a limitation. The results of our algorithm should provide a starting point for deep inspection of packets and flows.

4.3.3 Analysing Repositories

We made a prototype to analyse repositories using the algorithm from Section 4.3.2. We analysed two existing repositories using our prototype. The first repository we used was Simpleweb [26], maintained by the University of Twente. It was included here because it is

```
INIT usableFlows to {}        # all usable flows found
INIT testFlows to {}          # all flows found so far
FOR each packet in data file
    IF packet is SYN
        CREATE flow from packet
        ADD flow to testFlows
    ELSE IF packet belongs to flow in testFlows
        SET flow to flowOf(packet)
        ADD packet to flow
        IF packet is FIN or RST
            REMOVE flow from testFlows
                ADD flow to usableFlows
END FOR
INIT fakeGaps to {}
FOR each flow in usableFlows
    FOR each packet in flow
        IF packet is ACK
            IF acked packets are not in this flow
                ADD tuple (flow, packet) to fakeGaps
    END FOR
END FOR
```

Figure 9: Pseudo-code of the fake gap detection algorithm

publicly available and easily accessible. We analysed a subset of data covering all 6 locations the Simpleweb repository provides, totalling to 224 data files. The second repository is the WIDE traffic repository [27], maintained by the MAWI working group. It contains data files with traffic of several trans-Pacific lines. We used one data file from samplepoint A to test our prototype and we analysed 27 data files from samplepoint F, which were all over 1 GB in size. We chose samplepoint F since this one is daily updated while the other samplepoints are discontinued or were not accessible. The full table of results created by our prototype, and the prototype itself, can be found at [28]. Our prototype calculates fake gap statistics using various intervals. That is, fake gaps from different flows within this interval are grouped together and reported as a single fake gap.

Figure 10 shows the estimated number of missing packets per repository we tested and the estimated number of fake gaps within an interval of 0.05 seconds. The left part of the graph shows the extreme values. A first observation includes the presence of fake gaps in almost all tested data files from the WIDE repository. Also note that not all test data from analysing Simpleweb was plotted, instead we plotted the 28 highest values. It can be observed from the raw test data, that there is an absence of fake gaps in 50.8% of the tested data files of the Simpleweb repository. Data files from location 2 are almost solely responsible for this. A possible explanation could be the low amount of traffic at this location.

Figure 11 plots the percentage of affected usable flows that have at least one fake gap. For the WIDE repository, on average 0.49% of the usable flows is affected by at least

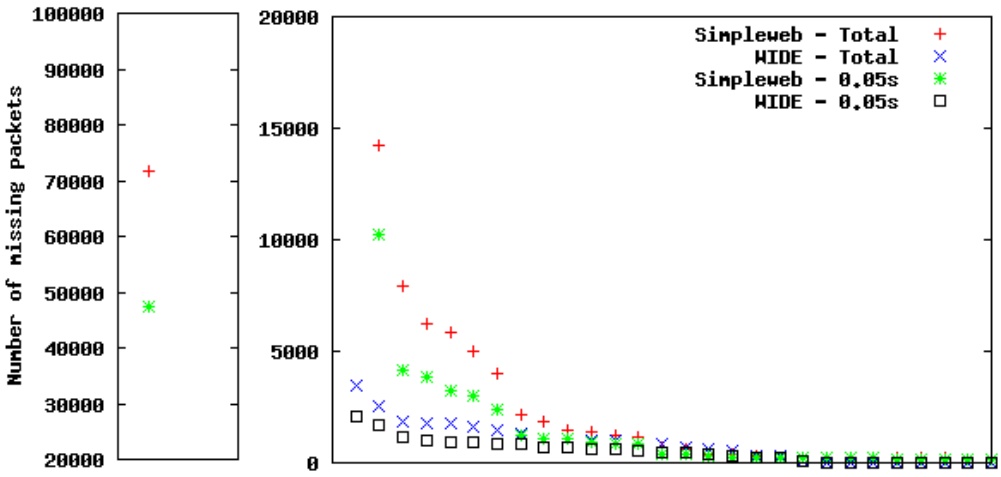


Figure 10: Estimated number of missing packets and fake gaps

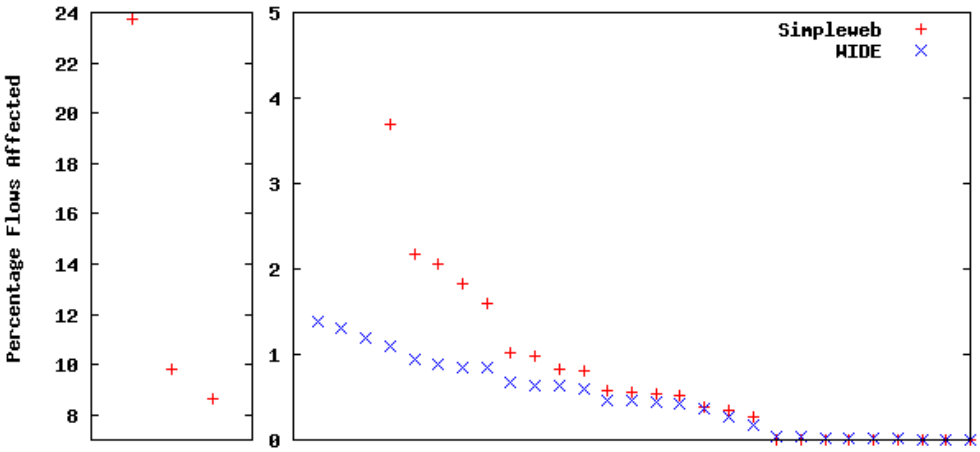


Figure 11: Percentage of affected usable flows

one fake gap. When ignoring data files without fake gaps, the average percentage of affected usable flows is 0.51%. Of the tested data files of the Simpleweb repository, an average of 0.27% of the usable flows in all data files was affected by at least one fake gap. When ignoring consistent files, the average is 0.55%. The highest value is in the file loc2-20030718-1530, where 23.72% of the flows is affected by at least one fake gap.

4.3.4 Conclusions

We have described our research on the consistency of network traffic repositories. Before answering the main research question, we first look at two sub questions identified in Section 4.3.1.

The first sub question was *"How can we detect fake gaps?"*. We proposed an algorithm in Section 4.3.2, which extracts TCP flows. Then, it tries to identify fake gaps, packets that

are not recorded by the recording device but were sent. The algorithm checks whether all data in the TCP flow is present by analysing TCP headers.

The last sub question, *"How consistent are today's repositories?"*, was answered in Section 4.3.3. We performed measurements on the Simpleweb [26] and WIDE [27] repositories. We showed both repositories contain inconsistencies. In the Simpleweb repository an average of 0.27% of the investigated TCP flows was affected by at least one fake gap. For the WIDE repository, this average was 0.49%. The research covered a substantial subset of data from both repositories. We analysed 28 data files from the WIDE repository and 224 data files from the Simpleweb repository.

Going back to the main research question, *"How can inconsistency be detected in a TCP traffic repository?"*, we can now conclude detecting inconsistency is possible by using the proposed algorithm, which detects fake gaps. The knowledge that a repository is not always consistent is very important for research where it is critical to have all data recorded, like research on packet loss. For this kind of research, it is recommended to take possible inconsistency in the repository into account and, if no statistics are present, analyse the repository data before using it.

Future research could include extending the proposed algorithm to support TCP flows that are not completely present in the data file. This research can be used together with algorithms like the one described in [25], which checks for anomalies in traffic rate, to find the exact locations of fake gaps. This can, in turn, be used to draw conclusions about non-TCP traffic, thereby getting a better overview of the consistency of a network traffic repository.

4.4 perfSONAR Tools Evaluation

The goal of this activity was to evaluate perfSONAR NetFlow tools for flow collection solution and assess its applicability to easily subscribe and request different NetFlow sources and store them and/or present to the user.

Section 4.4.1 provides an overview of perfSONAR architecture. Section 4.4.2 describes testing of Flow Subscription Measurement Point to request near real-time streams of flow packets and section 4.4.3 describes testing of Flow Selection and Aggregation Measurement Archive to perform remote flow selection and aggregation requests.

4.4.1 perfSONAR overview

perfSONAR (Performance focused Service Oriented Network monitoring ARchitecture) [29] is a result of GN2 and GN3 [30] EU-funded projects and aims at providing a framework for performing multidomain measurements. It is a result of international collaboration and is deployed in the European Research Network GEANT and the connected National Research and Education Networks (NRENs) as well as selected international projects, e.g., LHCOPN. The name reflects the choice of a Service Oriented Architecture for the system implementation. The architecture of this monitoring framework is depicted in Fig. 12.

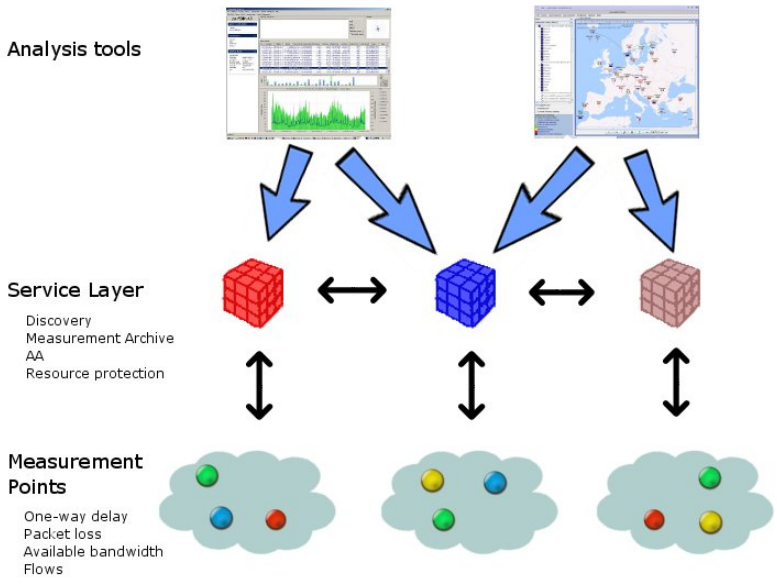


Figure 12: perfSONAR architecture

The Measurement Points are the lowest layer in the system and are responsible for measuring and storing network metrics as well as for providing basic network information. The measurements can be carried out by active or passive monitoring techniques. The Measurement Point layer of a domain consists of different monitoring components or agents deployed within the domain. A monitoring agent provides information on a specific metric (e.g. one-way delay, jitter, loss, available bandwidth, NetFlow data) by accessing the

corresponding Measurement Points. Each network domain can, in principle, deploy Measurement Points of its choice. The Service Layer is the middle layer of the system and consists of administrative domains. It allows for the exchange of measurement data and management information between domains. In each domain, a set of entities (services) is responsible for the domain control. Each of them is in charge of a specific functionality, like authentication and authorization or discovery of the other entities providing specific functionalities. In particular, the Measurement Archive Service (MA) is designed as a repository for measurement results. The interaction of the entities inside a domain as well as other domains is using the perfSONAR protocol and may not be visible to the end user. Some of the entities contain an interface which can be accessed by the User Interface layer. This layer consists of visualization and analysis tools (user interfaces) which adapt the presentation of performance data to be appropriate for the needs of specific user groups. In addition, they may allow users to perform tests using the lower layers of the framework.

4.4.2 Flow Subscription Measurement Point

The Flow Subscription MP [31] is a Java application developed by SURFNet which makes it possible to request near real-time streams of flow packets (that is Netflow or Sflow exported by routers), as if they were coming directly from the routers where the information originated. This allows clients of this perfSONAR service to subscribe to flow information from different locations and still use their own favorite flow collector and processing tools.

The MP collects one or more NetFlow streams and clients of this service. Users can specify the router(s) from which they want to receive flow information, and can further tune the amount of information sent by creating a filter. As flow information can be privacy-sensitive, the Flow Subscription MP can anonymize the IP addresses before the information is sent to the client. In addition, when setting up a NetFlow stream between the MP and a client the flow information is sent through an encrypted tunnel to protect the information. Zebedee [32] software is used here to establish encrypted, compressed tunnel for data transfer and the only type of data that goes through perfSONAR layer is a subscription request to set up the tunnel and data stream, keepalives to maintain the data stream and unsubscribe requests to finish transmission. The flow data collected by the Flow Subscription service is replayed, anonymized (when configured) and send through the tunnel based on client subscriptions. On the client side the data exits the secure tunnel on a specific UDP port as it would be coming from the direct router stream and can be collected by any flow collector tool to store or process the NetFlow data. Parallel client connections can be established to request data from different NetFlow sources.

The MP provides its users with on-demand and real-time access to (a selection of) flow information for a specific amount of time, allowing them to perform their own security analysis, performance monitoring calculations or data collection using the tools of their choice.

Testing

The Flow Subscription MP is available as software package for both Debian GNU/Linux and RedHat distributions. We used .deb package and APT packaging tool in order to install the MP in Debian 4.0. This required adding perfSONAR repositories into the system

and installation of Java v5. The installation was very straightforward and installed all required dependencies like nfdump, Tomcat 5.5 and zebedee. After restarting Tomcat, the application was ready for configuration. Unfortunately after initial configuration it turned out that the packaged version had a bug preventing the software to correctly recognize NetFlow exporters. After contacting the former but still active) developer, we were given a new WAR file created from the SVN repository and it worked fine. For the clarity of this description the following steps are using this type of installation. The bug was reported and should be fixed.

For testing purposes we used two routers sending NetFlow v5 to the test server where the service was deployed:

- PIONIER NREN core router 212.191.126.4, using port 9001
- POZMAN MAN core router 150.254.163.165, using port 9002

Initial configuration of the Flow Subscription MP required specifying flow exporters. Configuration of the service is done with a graphical interface called WebAdmin. This is a set of Web pages for basic and advanced configuration. It is not required to configure all of the options in order to run the service - some of them are perfSONAR specific (like registration to directory service). The only ones required at the beginning for our purposes were flow exporters data and WebAdmin password change. The rest could be left default. The configuration page was available under `http://loco2:8180/ps-mdm-flowsub-mp/` while the web service itself was available under `http://loco2:8180/ps-mdm-flowsub-mp/services/FlowsubscriptionMeasurementPointService`. Here we found a difference between the installation guide available in the Web, which suggested a wrong service URL and an exact service URL. This information was corrected based on our test. Fig. 13 depicts the configuration of the two aforementioned exporters. The last configuration point at the service side was to change default security settings. The Flow Subscription MP has been shipped with a filter configured by default and we had to change it in `/var/lib/tomcat5.5/webapps/surfnet_java-flowsubscription-mp/WEB-INF/web.xml` in order to allow client access.

In order to run the client we used one of the EmanicsLab servers, where missing zebedee, python and SOAPpy module were installed. Then we downloaded the python client directly from the perfSONAR repository. The client receives NetFlow data through an encrypted tunnel by subscribing to the MP. The only data that goes through the perfSONAR layer are the control messages (web service requests and responses). Before tests started the client had to be configured. This required us to enter the Flow Subscription MP URL, IP address of the client, port we want to receive data (7777) and a name of the NetFlow source (as configured at the MP side) we wanted to subscribed NetFlow data from. The client currently supports subscription to a single NetFlow stream. By default the service also uses the Crypto-PAn module with 32 characters key to anonymize IP addresses. Fig. 14 shows the output of the process running the Python client. It starts with subscribing to the MP service and then informs us about establishing the zebedee tunnel. Then it periodically reestablishes it in order to receive flow data. When the client terminates, it unsubscribes from the service.

During our tests we successfully subscribed at the EmanicsLab server to our NetFlow streams from the Flow Subscription MP and received NetFlow packets from aforemen-

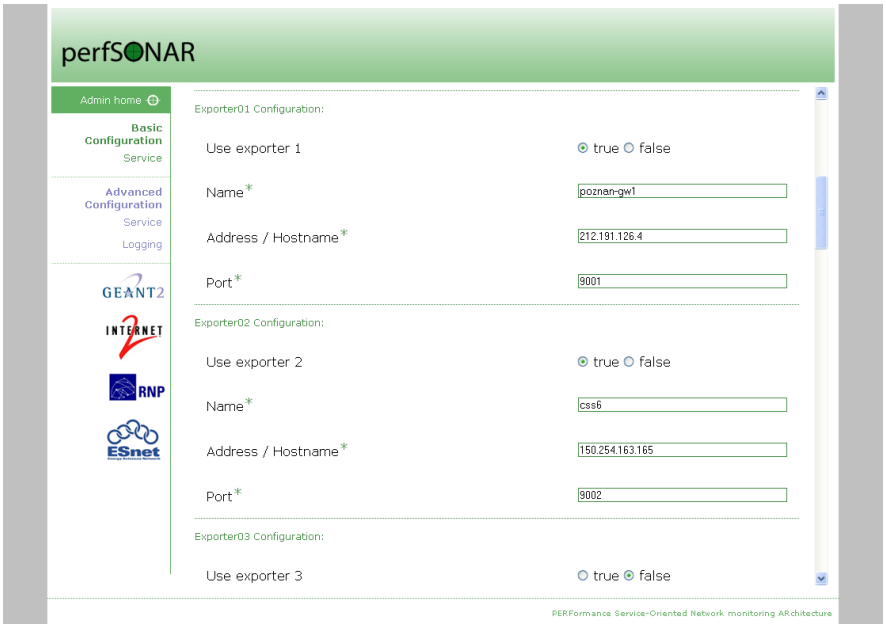


Figure 13: FlowSub MP administration interface

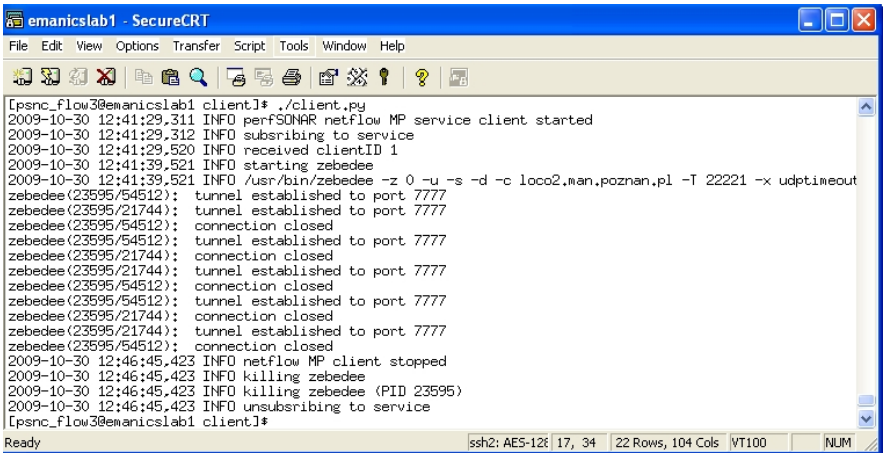


Figure 14: Running subscription client

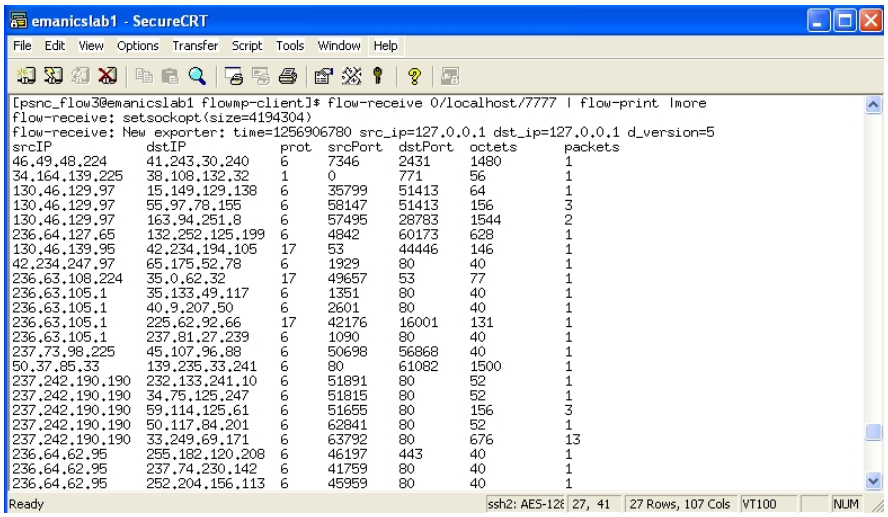


Figure 15: NetFlow data received at the given port

tioned routers with the use of a zebedee tunnel. Fig. 15 shows NetFlow data from packets received on port 7777 using the client. Then the NetFlow packets could be easily locally processed by any NetFlow analysis tools.

4.4.3 Flow Selection and Aggregation Measurement Archive

The Flow Selection and Aggregation MA [33], developed in Java by SURFNet, acts as a perfSONAR interface wrapper around the functionality of the flow capturing and processing commonly used tool nfdump [34]. This makes it possible to perform remote flow selection and aggregation requests. The MA is accompanied by a plugin for perfSONAR visualization tool perfsonarUI [35] that gives access to all available functionality. The perfSONAR community uses this dedicated analysis application perfsonarUI - Java Web Start graphical user interface client for querying a range of perfSONAR services deployed around the world. The latest version is available here: `http://perfsonar.acad.bg/psui_beta/perfsonar.jnlp`. With the Flow Selection and Aggregation MA service users can perform perfSONAR-based remote nfdump style selection and aggregation queries on stored log files. This enables them to search for flows patterns, security related information, and to debug network related problems. Depending on the storage capacity of the MA, queries can be performed on flow information from weeks or even months ago. The MA supports the following types of data requests:

- flowStatistics return statistical information about the selected flows from a given group of routers
- rawFlows return specific fields from the selected flows. An optional nfdump style filter rule can be used to limit the number of matching flows and/or an aggregation rule can be used to accumulate information
- topFlows return the top N flows given a nfdump style top filter rule

Testing

The Flow Selection and Aggregation MA is available as software package for both Debian GNU/Linux and RedHat distributions. We used .deb package and APT packaging tool in order to install the MP in Debian 4.0. This required adding perfSONAR repositories into the system and installation of Java v5. The installation was very straightforward and installed all required dependencies like Tomcat 5.5. After restarting Tomcat, the application was ready for configuration.

For testing purposes, we used a router sending NetFlow v5 to the test server where the service was deployed:

- PIONIER NREN core router 212.191.126.4, using port 9001

Then for test purposes, these NetFlow packets were stored locally by the nfcapd service, run with the script attached to the installation package and installed in /etc/init.d folder.

Initial configuration of the Flow Selection and Aggregation MA required specifying flow exporters. Configuration of the service is done with the graphical interface called WebAdmin. This is a set of Web pages for basic and advanced configuration. It is not required to configure all of the options in order to run the service - some of them are perfSONAR specific (like registration to directory service). The only ones required at the beginning for our purposes were flow exporters data and WebAdmin password change. The rest could be left default. The configuration page was available under `http://loco2:8180/ps-mdm-flowsa-ma/` while the web service itself was available under `http://loco2:8180/ps-mdm-flowsa-ma/services/FlowsaMeasurementArchiveService`. Here we again found a difference between the installation guide available in the Web, which suggested a wrong service URL and the exact service URL. This information was corrected based on our test. Fig. 16 depicts the configuration of the exporter we used in our tests. The last configuration point at the service side was to change default the security settings. The Flow Subscription MP has been shipped with a filter configured by default and we had to change it in `/var/lib/tomcat5.5/webapps/surfnet_java-flowsubscription-mp/WEB-INF/web.xml` in order to allow client access.

After initial configuration, we might have started retrieving data from the service using perfsonarUI. Of course, it is possible to query the service with other tools provided they conform to the perfSONAR protocol. The user is constructing a query with the client. The client then transmits this query to the service and the web service transforms the query into nfdump commands. These nfdump commands have Netflow data or statistical data as a result and are converted back to NMWG[3] XML (used by the perfSONAR protocol), and sent to the client where they are visualized. Before the tests started, a new configuration file was created in order to specify Flow Selection and Aggregation MA service URL. The configuration file is read then by the dedicated tab in the GUI. Unfortunately, it turned out that the Selection and Aggregation MA tab in perfsonarUI is not working as expected and is missing some important features. For example, the ”Options” configuration window does not currently enable to specify the IP address or DNS name of routers, which is crucial to retrieve the data. To avoid this problem, the router’s name was entered directly into the ”Settings” window. Other fields of the ”Option” window work well. The other problem we

[3]Network Measurement Working Group of the Open Grid Forum

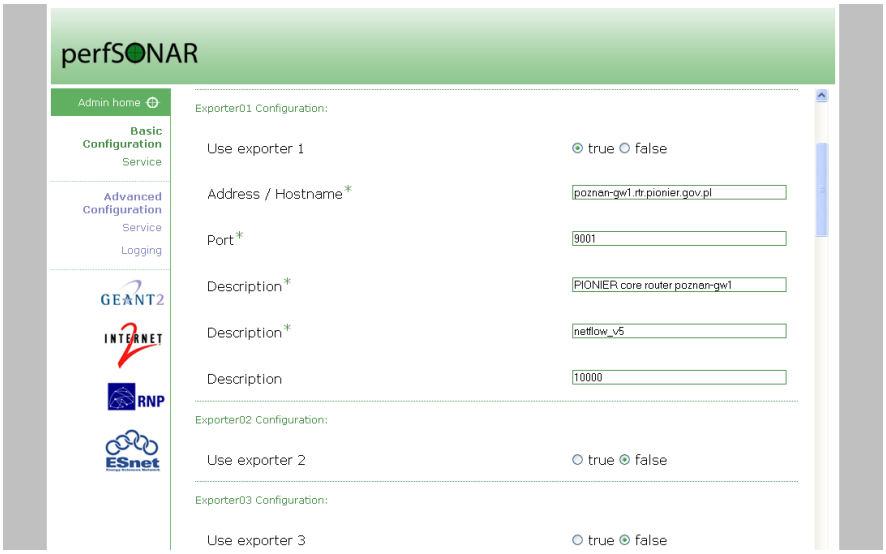


Figure 16: FlowSA MA administration interface

noticed is lack of anonymization on the service side, which may prevent users in a shared usage of the tool. These problems were reported to perfsonarUI developers.

First we tested different TopN requests as the most useful command. We choose the analysis period and then additional parameters like type of statistics (record, IP addresses, ports, AS numbers, interface) and the order by which the statistics is ordered (flows, packets, bytes, etc). Number of statistics was also specified. Fig. 17 depicts the result of an example query using the perfsonarUI using Record type of statistics, aggregating by Packets and displaying 10 results where a few fields are displayed providing information about start time of the flow first seen, duration of the flow, flow details, bytes transferred, number of packets and protocol type.

Then we tested getting statistical information about flows collected by this perfSONAR service within a specified period of time. Therefore no options are configurable apart from time period. Fig. 18 depicts the result of an example statistical query to Flow Selection and Aggregation MA using the perfsonarUI. The result provides information about total number of flows, packets, bytes and the same information but per protocol, start of the first flow and end of the last flow.

Finally we tested raw flows which could be used for detailed display of each NetFlow record and could be rather rarely used. In the Options window we selected Raw type of query and a filter to limit the amount of data (port 8080). We also chose fields to be displayed (source and destination IP address, source and destination port number, source and destination AS number and number of bytes and packets. Fig. 19 depicts the result of such raw query to Flow Selection and Aggregation MA using the perfsonarUI. If the time range is too wide or no specific filter is applied, it results in a lot of raw data and generates an error about too many results retrieved.

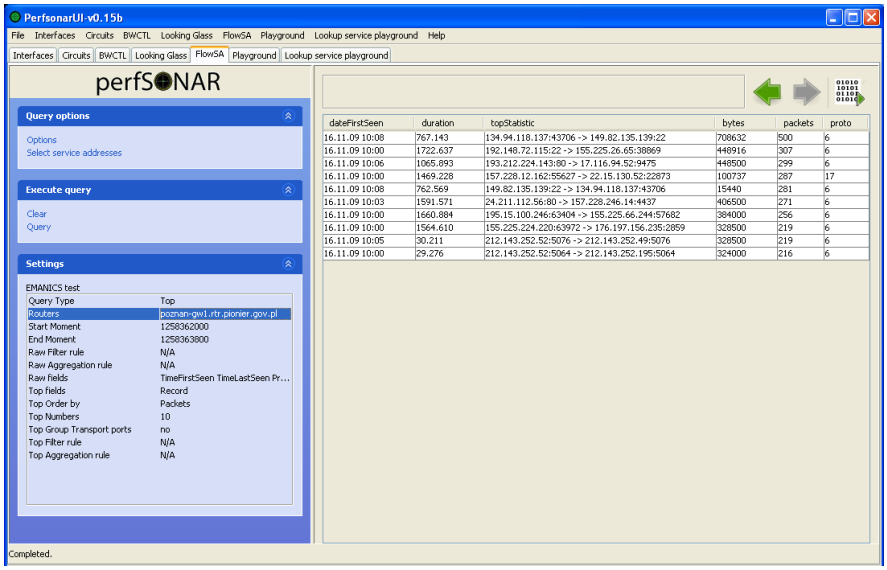


Figure 17: perfsonarUI accessing Flow Selection and Aggregation MA - TopN flows

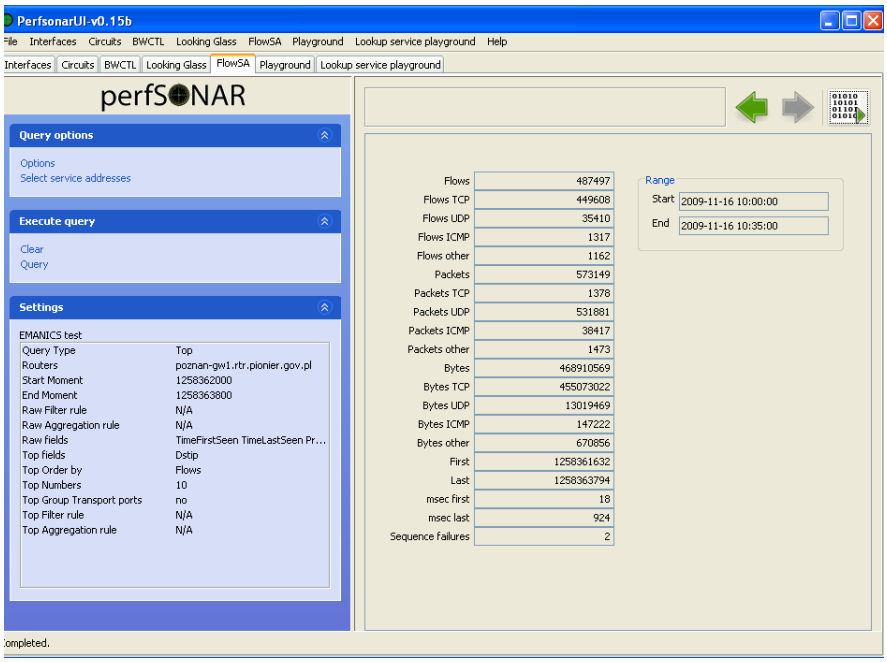


Figure 18: perfsonarUI accessing Flow Selection and Aggregation MA - statistical information about flows

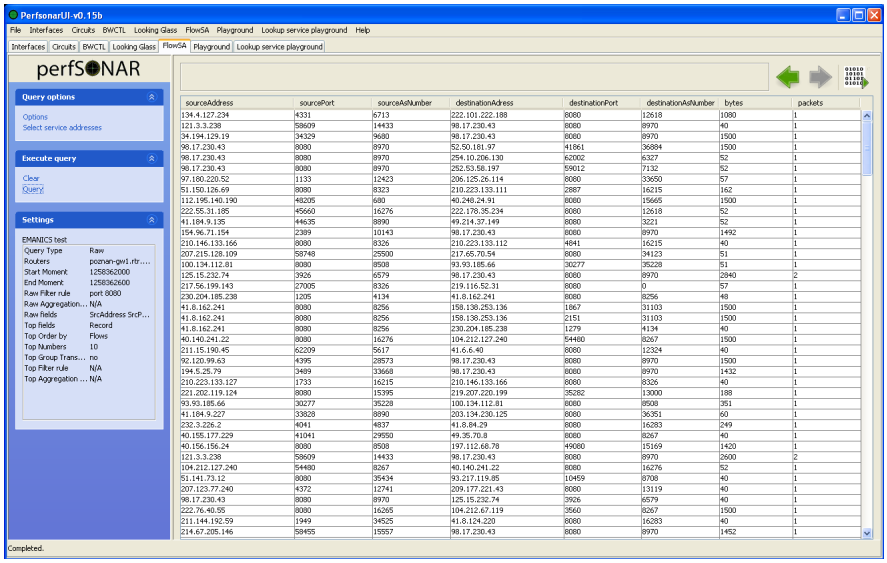


Figure 19: perfsonarUI accessing Flow Selection and Aggregation MA - raw flows

4.4.4 Conclusions

We tested two perfSONAR tools: Flow Subscription Measurement Point to request near real-time streams of flow packets and Flow Selection and Aggregation Measurement Archive. Both are using the perfSONAR architecture as a basis. This European activity seems to be well established in a research community with the ongoing GEANT project, which continues working on perfSONAR deployment, thus increasing the community of potential users. This should hopefully provide a long term support for the tools. There are also steps in order to standardize the perfSONAR protocol under Open Grid Forum (OGF). During the Flow Subscription MP tests, we found a few problems which were reported to the perfSONAR community and we hope will be taken into account during next releases. The configuration part was straightforward although the client is very simple and not sophisticated. We think the service is a useful tool in order to publish NetFlow data in a secure and controlled way especially in the management and monitoring of networks in a multidomain environment or collaboration of distributed partners. Especially for those networks, which already use perfSONAR services in order to share other monitoring data. The Flow Subscription MP may be used in order to give access to flows streams by simply subscribing to a selected device for a selected period of time necessary to collect data without the necessity to receive flows all the time. A potential problem could be the zebedee tool, which is in a good shape now but seems to be not developed anymore (the last stable and development versions 2.4.1A are from 2005/09/06). The results of Flow Selection and Aggregation Measurement Archive tests were not such optimistic mainly due to analysis tool problems, which currently may prevent users from using this set of perfSONAR products. We could not properly configure the GUI to access the service. It is obvious that these perfSONAR web services needs bug corrections in order to make them usable. The graphical interface enabling access to the data through perfSONAR interface also needs a few corrections and enhancements that are necessary to make it easy to use and valuable to the community. We submitted appropriate bugs and we hope that the feedback from our tests will help the developers to enhance perfSONAR tools.

5 Trace Visualization and Anonymization Tools

5.1 Trace Anonymization

Network traces contain sensitive data, such as for example SNMP community strings or even full communication transcripts in case of full packet traces. Even basic things such as IP addresses can be considered sensitive data and thus should be handled with extreme care, especially when sharing network traces between different organizations. Various anonymization techniques exist in order to remove such data or alter it in a way that makes further trace analysis possible without revealing information about data sources or the data itself. Whether the anonymization is effective enough can be verified by investigating possibilities of such trace analysis that can reveal information about the origin of anonymized traces.

5.1.1 Introduction

Collecting network traces is essential to give researchers tools to fully comprehend the workings of nowadays complex network infrastructures. Network traces contain valuable information in terms of network traffic characteristics, network management schemes in use and network architecture itself. Analysis of this data gives even greater benefits for the networking community when it is based on big data sets collected across various different domains, managed by different organization, varying in nature of the traffic served (for example, a university network will obviously have different traffic characteristics than a local area network in a corporate environment etc.). Collecting such traces, however, introduces several problems related to data confidentiality. Raw network traces, especially full packet traces, may contain sensitive information that should not be made available to anyone without interested parties' consent. This information may include confidential communication such as clear text passwords, unencrypted instant messaging conversations, e-mail, IRC sessions or any other type of electronic communication protected by privacy laws. Obviously, this kind of information must be removed from traces or in case the removal would change analyzed trace parameters significantly, it must be anonymized, so that the original content becomes unrecoverable. Moreover, even information that characterizes the basic network information flow, such as IP addresses, may be considered confidential as a basic analysis of network traces containing such unmodified information may lead to revealing the topology of the network the traces originate from and many other important details, such as services run on particular machines (which gives potential attackers basic clues as to which hosts are most vulnerable and may serve as entry points into the network in case of the attack). Protocols carrying management information, while not containing anything that might be considered private communications, contain information crucial to secure operation of the network itself, such as community strings in case of the SNMP protocols or other sensitive data that might serve as a tool in a successful hacking attempt. All those examples explain the necessity for strong anonymization procedures to be followed before releasing trace data sets for analysis by third-party researchers.

Anonymization procedures vary depending on the nature of anonymized data, of course. And even for the same data, various anonymization schemes may be in use in order to

preserve some desired characteristics of the original trace (for example prefix preserving anonymization), for the benefit of greater anonymity. Exactly how secure is a given scheme may be described by analyzing how hard it is to break such anonymization. Several attack scenarios have been proposed by researchers to demonstrate vulnerabilities of supposedly securely anonymized traces. Exploiting such weakness may endanger the original network that released the data in question in various ways from revealing the network architecture to revealing critical system information, such as for example SNMP community strings, used for network management authentication.

5.1.2 Attack Scenarios

Several attack scenarios were identified as potentially dangerous threats to anonymization techniques. Each approach gives the attacker slightly different possibilities of extracting more information from the traces than intended by the institution that released the data. Nevertheless protection against as many of these attack techniques as possible is necessary to encourage releasing traces into public to allow for a community-wide analysis.

- Infection attacks

  In this attack method, the attacker injects information known to him into the data stream that is logged by the organization responsible for providing traces. Knowing the exact data that was sent through the network, the attacker may undertake an analysis effort in order to try to detect his/her original sequences inside the already anonymized data traces. Since the attacker knows where to expect the data and its original form, a further analysis may reveal the anonymization scheme used for this particular data set.

- Fingerprinting attacks

  Analysis of trace data may reveal a specific role of particular nodes within the original network, for example detecting a web server by the amount of traffic exchanged through port 80. While the actual IP addresses of such nodes are not explicitly known, the attacker may try matching such behavioral characteristics and relations between nodes in the anonymized traces to the already known architecture of some networks, thus detecting the original sources of the traces.

- Structure recognition

  Related to fingerprinting attacks, discovering relations between anonymized objects may lead to mapping those relations onto the source network, thus discovering its structure.

- Known mapping

  Once the attacker knows some mapping between anonymized and non-anonymized hosts, an analysis may reveal further information about nodes in the source network.

- Cryptographic attacks

  Although unlikely (since cryptographic algorithms used in various anonymization tools are widely known and are already well-established), this is another potential

weak point of the anonymization procedure. Encrypted data can theoretically be exposed if the algorithm can be broken.

Each attack scheme may be more or less effective against traces anonymized with different tools. Those are chosen depending on the nature of confidential data and the desired level of confidentiality.

5.1.3 Anonymization tools

Various anonymization tools may be used to process confidential information in standard IP protocol fields as well as in the payload, depending on the protocol being used [36]. Several examples of tools designed to handle different protocols are presented below.

- AnonTool

  This software is designed especially with NetFlow in mind. Developed as an implementation of the Anonymization API (AAPI), the tool provides mechanisms for anonymization of most NetFlow fields (ensuring the resulting NetFlow contains correct checksum). In case of IP address anonymization, the techniques used include prefix-preserving anonymization, removal of IP address, partial masking of the address etc. [37].

- TCPurify

  Designed as a tool similar to the widely known tcpdump, but much less complex, TCPurify focuses on retrieving IP headers only, destroying payload in most cases completely. IPv4 addresses are anonymized during a packet capturing process. The algorithm in use allows address randomization reversible using an information file generated at capture time [38].

- Tcpdpriv

  A tool for general anonymization of IPv4 traces. Tcpdpriv destroys the payload and allows for more complex address anonymization, including the prefix-preserving anonymization. Other IP fields may be anonymized or removed from the traces [39].

- SNMPDUMP

  The `snmpdump` tool specializes in the transformation and anonymization of SNMP traces, which is particularly critical as such traces often carry information critical to network security. The IP addresses are anonymized using a prefix-preserving algorithm, port numbers are anonymized as well. SNMP community strings (that are presented in clear text) are removed to prevent unauthorized access to monitored and managed devices [40].

- CANINE and its successor FLAIM

  CANINE was originally developed as a tool for converting traces between multiple NetFlow formats and anonymizing several fields: IP addresses, timestamp, port numbers, protocol number and byte count using several methods, such as prefix-preserving IP anonymization (simpler algorithms such as random permutation or

truncation were also available). CANINE was originally a Java tool, therefore processing speed was a concern. The project evolved into a faster (rewritten in C++) FLAIM framework that is more flexible, general and easily extensible. The engine offers complex anonymization techniques for a wide variety of log and trace files [41, 42].

- Tcpmkpub

  Accepting pcap files as its input, Tcpmkpub performs IPv4 prefix-preserving address anonymization (using the Crypto-PAn algorithm) as well as removal of potentially sensitive data from packet payload. It is also able to perform anonymization on Ethernet addresses. This tool is considered particularly useful for edge networks [43].

Depending on anonymization options offered by the software of choice (and level of confidentiality desired by trace source), the resulting network traces are susceptible to different attack schemes. This relation is explained below. Due to low availability of deanonymization tools, the work focuses on theoretical analysis.

5.1.4 Attack Scenarios and Prevention

When planning a release of network traces into public, possible threats to confidentiality of network users data and to the network integrity must be evaluated and selected anonymization options must be selected adequately to the site's privacy policy and other telecommunication privacy laws. In order to perform the anonymization correctly, one needs to consider possible deanonymization attack scenarios and choose anonymization tools that fit the most the nature of the data and offers best anonymization algorithms.

Some theoretical attack scenarios were defined in the literature already [44]. This article is focused on scenarios presented below since these are the scenarios most likely to occur in case of traces collected by EMANICS partners.

- Attacks against anonymized traces of unknown origin

  If all the attacker has access to is an anonymized network trace without any information on the original source of the data, the attack must be based entirely on the analysis of the trace contents, since there is obviously no possibility of influencing the trace by engaging techniques such as active fingerprinting [44]. If IP address anonymization is performed using most simple algorithms such as partitioning (assigning IP addresses to one per each of several defined partitions instead of explicitly listing every IP address in the trace; this is a case of many-to-one address mapping) or removal of IP addresses from the trace completely, there is no way of extracting information about original network structure accurately enough so it can be useful for any possible further attack. The problem is, traces anonymized this way loose important properties from the point of view of analysts working on the data. Prefix-preserving anonymization is often used in cases where identification of particular hosts is essential. Such an anonymization technique is a case of one-to-one IP addresses mapping, where each IP address is mapped onto another one calculated in

such way that if given 2 addresses share an n-bit prefix before anonymization, they will share an n-bit prefix after the transformation. While the complexity of the algorithm makes it virtually impossible to reverse the anonymization itself, such traces are susceptible to attacks based on statistical trace analysis.

- Attacks against anonymized traces from known source networks

  If the trace contains IP addresses anonymized using one of the one-to-one mapping methods (such as previously mentioned prefix-preserving anonymization or IP address permutation etc.), the attacker may attempt to reconstruct the original network structure. It is possible to do so using statistical analysis of network traffic directed to particular IP addresses. For example, the traffic of web servers is different in terms of flow duration and number of connections served per time unit from the traffic directed to a personal computer connected to a peer-to-peer network. By identifying hosts related to such typical network traffic characteristics, an attacker can try to identify the role of each node. If the original trace source is known to the attacker, discovering the roles may reveal a mapping pattern that was used during the anonymization procedure, which is particularly dangerous in case of simple address permutation mappings. In such a case, the deanonymization attack may lead to a total deanonymization of IP addresses throughout the trace. Prefix-preservation anonymization algorithms (or any anonymization schemes that are based on more complicated mappings) are not as badly affected. Nevertheless, revealing identities of particular hosts in the anonymized trace may affect the data security, especially if packet payload remains unanonymized (as it may contain data non-critical itself, but dangerous when associated with a particular address, such as plain text passwords etc.), which is not uncommon in traces due to confidentiality laws imposed on electronic communication.

- Attack during a trace collection activity

  If the trace collection activity is in progress at a given site and the attacker has access of any kind to the network, he/she can use the aforementioned active fingerprinting that involves injecting data into the analyzed network. Knowing the IP address of, for example, a web server the attacker may request content from the server in a particular way, creating a network traffic pattern that can later be recognized in the anonymized traces. Such requests can be identified, for example, by flow sizes, specific flags inside packet headers or other characteristics that the attacker expects to remain unchanged after the anonymization process. Identifying those requests in the released trace may result in revealing the anonymization mapping (the severity of this problem varies depending on algorithm used). Combined with similar fingerprinting activity against other hosts, this may lead to revealing communication patterns between selected nodes in the network.

Attackers obviously can extract the same information from anonymized traces as researchers do. One cannot eliminate the risk of deanonymization of traces without changing the traces' properties significantly. However, the risk may be limited by carefully selecting the anonymization tool for each purpose and keeping appropriate trace sharing policies. IP anonymization has been quite well developed over time and tools are available capable of anonymizing IP addresses using complex algorithms so it becomes virtually irreversible.

Statistical attacks seem to be a bigger problem since they mostly depend on the very same data that is essential to network management research. Altering statistical trace properties makes the whole analysis pointless since it would no longer resemble original network traffic in any way. Because of this, it is still essential to release traces under non disclosure agreements between interested parties. Otherwise, security-critical (the exact definition of the word critical depending on particular traces nature) information should be removed from the trace completely. This should limit attackers opportunities to exploit anonymization weaknesses.

5.2 Visualization of Node Interaction Dynamics in Network Traces

Most network trace analysis tools provide graphical displays of traffic over time, showing traffic breakdowns in different time resolutions. While useful to get a first overview, such traffic plots are not sufficient in order to develop a deeper understanding of the exchanges captured in a network trace. For understanding certain traces, it is essential to develop an understanding of the interaction dynamics. A simple example are network or port scans where the scanning strategy reveals some insights about the tools an attacker might have used.

Another example are traces covering a specific type of traffic that is expected to exhibit a certain behaviour. Some of our previous work has been related to the collection and analysis of network management traces, and in particular SNMP traces [45, 46]. Initial results were published in [47] where we showed some simple static visualizations of the SNMP topologies discovered in our traces. While the simple static visualizations proved to be useful (one trace showed some unexpected anomalies due to dynamic address assignments), the static topology visualizations do not provide insights about the interaction dynamics. In particular, it is not possible to determine whether a data collection engine spreads the data retrieval traffic over a polling cycle or sends a burst of polling requests at the start of each cycle. Furthermore, it is not possible to see whether there are topology changes or if there are patterns of topology changes. (Note that topology changes on the management plane are usually caused by devices or links failing or returning back to their normal operational state.) To address these questions, the following experiments visualize the node interaction dynamics in a way such that it is possible to observe a pattern on a trace spanning days, even though the messages are exchanged with a round-trip time measured in the order of microseconds [48].

Note that the following description of two visualization experiments has been also published in [48].

5.2.1 Experiment #1: NAM

In our first experiment, we visualize the exchange of SNMP messages between managers and agents in order to highlight how SNMP monitoring engines distribute the polling load over time. While doing the conceptual design of the trace visualizer, it became apparent that *what* is being displayed should be decoupled from the *how* to display it. For describing what should be displayed we had to write our own tools because the SNMP trace format is relatively new and there are no powerful processing tools readily available yet. The comma separated values (CSV) SNMP trace format defined in [45] is very easy to parse in almost any programming language. Once the traces are parsed the flexibility of the programming language allows for any filter to be described and any output format to be used. We have used the programming language Python [49] in our experiment for it's rapid prototyping features. For visualization, we needed a tool that uses an open format and is able to display network topologies (graphs) and messages traversing the topology. We have chosen the popular network animator `nam` due to its ability to visualize the exchange of messages between nodes and our familiarity with this tool.

We first describe the network animator used for the experiment. We then describe how

we convert SNMP traces into input files for the network animator before we present and discuss the visualization results.

Description of `nam`: The network animator[4] (`nam`) is distributed as part of the `ns2` simulation software package. It is mainly used to visualize traces generated by the ns2 simulator. The `nam` tool is implemented in a mixture of C++, Tcl/Tk [50], and an object-oriented extension of Tcl. In general, the animation software is not easy to modify due to the high learning curve involved in understanding the interplay of the different programming languages involved.

The network animator reads a simulation trace file (also called a `nam` trace file), computes a graph layout for the recorded network topology, and afterwards it animates the messages passing over the links connecting the nodes. The graphical user interface provides controls to pause the animation, to change the animation speed, to zoom in and out, and to recalculate the graph layout or to manually reposition nodes.

The `nam` input file uses a relatively simple textual line-oriented format. We only describe a subset of the features here that we have used in our experiment. A `nam` trace file starts with a header containing version information, the list of nodes, and the list of links connecting the nodes. The version line has the following format:

```
V -t * -v 1.0a5 -a 0
```

The character `V` indicates that this is a version definition and the version number `1.0a5` is passed as the argument to the `-v` option. Nodes are defined using lines of the following format:

```
n -t * -a %d -s %d -S UP -v circle -c black -i black
```

The character `n` indicates that this is a node definition. The `-t *` parameter defines that this event does not have a time attached. The `-a` and `-s` parameters define the address and the id of the node. The `-S` parameter defines the node status while the `-v` parameter specifies the shape of the node and the `-c` and `-i` parameters specify the color of the node. Links connecting nodes are defined using lines of the following format:

```
l -t * -s %d -d %d -S UP -r %d -D %f -c black
```

The character `l` indicates that this is a link definition while the `-t *` parameter again indicates that this definition does not have a time attached. The `-s` and `-d` parameters specify the id of the source and destination nodes. The `-S` parameter defines the link status and the `-c` parameter the link color. The `-r` and `-D` parameters specify the data transmission rate of the link and the delay associated with the link. Finally, the color header lines specify the color coding of message types:

```
c -t * -i %d -n %s
```

[4] `http://isi.edu/nsnam/nam/`

The character `c` indicates that this is a color definition. The parameter `-t *` again indicates the time of the event. The parameter `-i` specifies the id of the color while the `-n` parameter carries a color name (according to the X11 color database). Following the `nam` file header, network message events are encoded using the following format:

```
h -t %f -s %d -d %d -e %d -c %s -i %d -a %d
```

The character `h` indicates a hop event. The parameter `-t` specifies the time of the event while the parameters `-s` and `-d` specify the source and destination node id of the message. The parameter `-e` carries the message size while the `-c` parameter carries a conversation identifier and `-i` a message id. The color of the message is specified using the `-a` parameter. Next to hop events, we use receive events. They have the same format in the `nam` trace file, except that the character `h` is replaced by the character `r`.

Conversion Algorithm for SNMP Traces: In order to use the network animator, we wrote a program to convert SNMP traces into the `nam` trace format. The program reads SNMP traces in the CSV format defined in [45]. In order to generate the `nam` header with the complete node and link list, our program needs to read the whole CSV file before `nam` output can be produced. In order to be able to share visualizations with other researchers and the public, we took care that our converter hides information such as IP addresses and absolute time-stamps.

For each transport address in the SNMP trace file, a node with a unique id is generated. The id is not derived from the transport address contained in the trace file. The links are inferred by determining all pairs of transport addresses that exchanged SNMP messages and translating this to source/destination node ids. For SNMP traces, the number of nodes and links encountered is usually small [47]. Finally, we generate for each SNMP message two nam trace events, namely a hop event and a receive event. The SNMP message events are color coded, allowing network analysts to distinguish easily between different request message types and responses.

An important issue is the time scale. Our visualization is designed to display traces covering weeks. However, in a real scenario, packets travel the network in fractions of a second. Any decent time scale that will allow a user to view the entire data-set will make the SNMP messages almost invisible. For this reason we tweak the network properties in the following way: we change the delay assigned to a link to 60 seconds. This is unrealistic, but it helps to make packets visible for longer time periods since it allows users to select faster than real-time timescales and still be able to see SNMP messages being sent. This change also helps to obtain a general overview over the network because the user actually sees all the data in the last 60 seconds. Another issue is the data rate of the links. It is used by the network animator to display the size of the message (in conjunction with the delay). For this reason, a 150 byte message on a 1Mbps link is too small to be displayed. Therefore we have also decreased the rate of the links down to 1kbps.

Results and Discussion: The generation of topology layouts is left to the network animator. Our conversion program does not provide any hints to the network animator concerning the graph layout.

Figure 20 shows a typical layout calculated by NAM. In this trace, two manager applications monitor an almost disjoint set of network devices. (Note that this layout is different

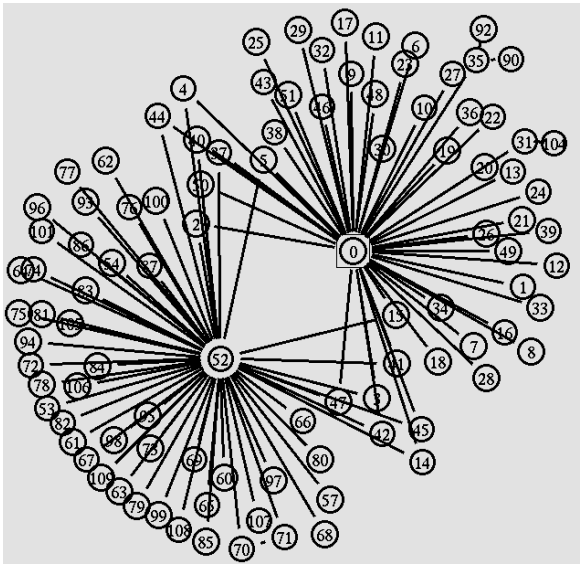
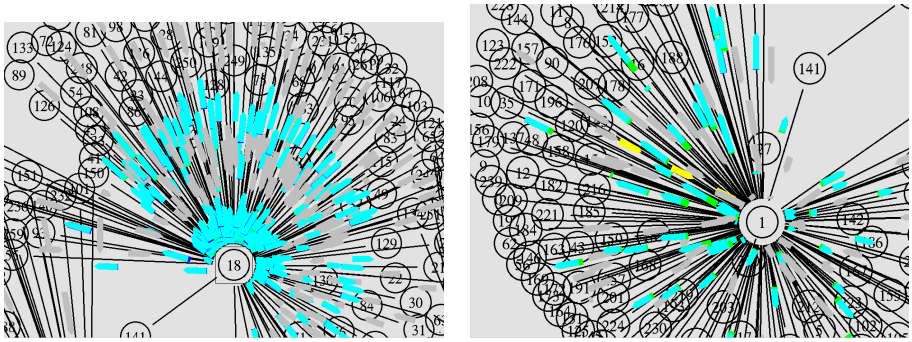


Figure 20: Network layout calculated by `nam`



(a) Message waves (b) Dispersed messages

Figure 21: SNMP interaction dynamics visualized in `nam` (trace l06t01)

from the layouts described in [47] since it is based on the network addresses of the nodes involved and does not distinguish different SNMP flows to the same endpoints.)

Figure 21 shows screenshots from a visualization of trace l06t01 (see [47] for more details about this trace). The left part (a) shows a manager polling devices by sending requests at about the same time to many devices. The devices respond with roughly the same delay causing subsequent requests to be sent out at roughly the same time as well. This leads to message “waves” in the visualization. The right part (b) shows a manager distributing the messages well over time, thereby avoiding bulky request/response waves. Part (b) also shows notification messages originating from one agent to inform the manager of some events.

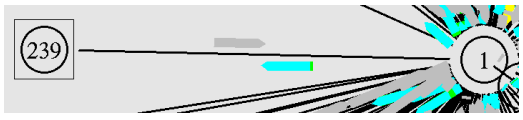


Figure 22: SNMP message interaction details

Figure 22 shows some details highlighting the visualization of request/response interactions over a single link. Due to the adaptation of the link properties (delay and data rate), a request and the corresponding response usually appear traveling on a link at the same time.

5.2.2 Experiment #2: NetViz/JUNG

In our second experiment, we aim at visualizing the changes of the manager / agent topology over time by calculating topology-change dynamics graphs. A topology-change dynamics graph is a undirected graph showing the relation between nodes based on the recorded activity. If a message between node X and node Y has been seen in the last t_{link} seconds, there is an edge between the nodes in the graph. If no activity has been seen on the link for a period of t_{link} seconds, the edge between the nodes is removed. Once a node remains with no active links attached to it (no edges on the graph), a timeout timer is fired and if no link attaches to it for t_{node} seconds, the node is removed from the graph. In general, one can expect that the relationship between SNMP managers and agents is rather static. Changes usually occur if there is a (notable) event in the network (devices powering up / down, notifications emitted that do not happen regularly). To understand a trace, it is therefore a good approach to find such topology changes and to further analyze them.

Description of NetViz/JUNG: In order to create topology-change dynamics graphs for traces that can be very large in size, we decided to take a two step approach similar to the way the network animator works. We have implemented a graph animation tool called NetViz that reads an ordered timestamped list of events as input (addition of a vertex, addition of an edge, removal of a vertex, removal of an edge) and runs the animation. An additional program is used to read trace files and to produce the intermediate file containing a timestamped list of events. One benefit of this approach (besides increased modularity) is that animation scripts are usually several orders of magnitude smaller than the original trace files they are derived from.

The animation file format is very simple. Each line has the following basic format:

```
<timestamp> <action> <identifier>
```

The `<timestamp>` of the first line contains an absolute timestamp while all subsequent timestamps are relative. The `<action>` is one of `Va`, `Vr`, `Ea`, `Er` (vertex add/remove, edge add/remove) and the `<identifier>` is a unique identifier for the vertex / edge to be added / removed.

For the visualization of the topology, we used the Java Universal Network / Graph Framework[5] (JUNG), a software library for the modeling, analysis, and visualization of data that can be represented as a graph or network [51]. The library supports a number of different graph layout algorithms, clustering algorithms, and user interface controls such as lenses. The library makes it easy to write Java [52] programs that dynamically update the graph by adding nodes and edges while the displayed layout is dynamically recomputed. We added a simple graphical user interface (GUI) with a few controls around the JUNG framework.

[5] `http://jung.sourceforge.net/`

Conversion Algorithm for SNMP Traces: The conversion program reads an SNMP trace in the CSV format defined in [45] and internally constructs a graph. Nodes are identified by their IP addresses and edges are identified by the IP addresses of the endpoints, independent of the direction of message exchanges. To each node and edge, there is a time-to-live counter attached, initialized with the values t_{node} and t_{edge}, passed as command line arguments. The timer for an edge starts running from the moment it is added to the graph, while the timer of a vertex starts running when there are no more edges connecting this vertex to other vertexes. The default value for t_{node} and t_{edge} was set to 600 seconds.

Results and Discussion: The graph visualizer reads the animation file and draws the graph as it changes over time. The placement of the nodes on the canvas is done according to a relaxation algorithm, which considers each edge as a spring and finds the node configuration with minimum potential energy. When vertexes or edges are inserted or removed, the new configuration is relaxed until it reaches a minimum again.

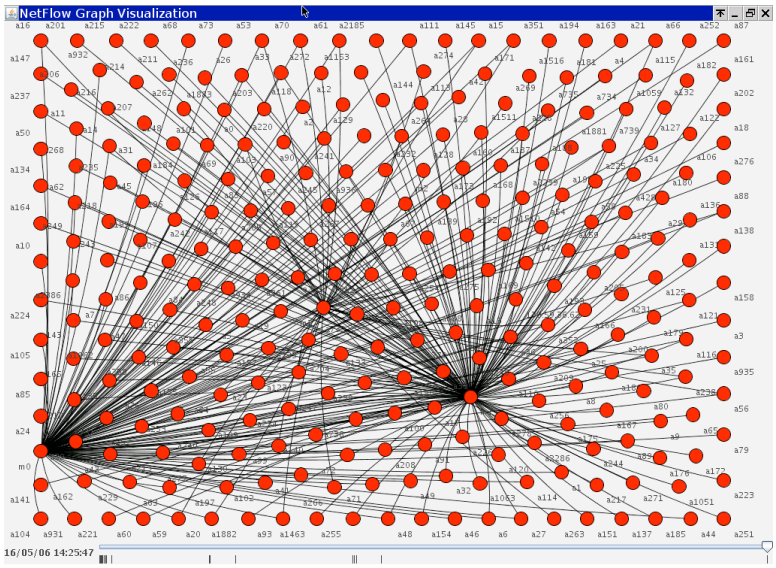


Figure 23: Dynamic network layout calculated by NetViz/JUNG (trace l06t01)

The GUI of the application, shown in Figure 23, provides a label with human readable time of the replay, based on the timestamps, which are in seconds since the Epoch, and a timeline with the regions of activity marked on it. Since in most of the cases the configuration of the graph remains intact for most of the time and just has several peaks of activity over the whole time period, running the animation at a constant speed is somewhat boring as nothing is happening most of the time. To deal with this, we introduced an idle speed at which the animation runs when there is no activity. Once it gets close to some activity, the animation switches back to normal speed, so that the user can see what changes exactly are taking place on the graph. The normal and idle speed can be passed as command line arguments, or otherwise default to 300 real-world seconds per 1 second animation time for normal speed and 6000 real-world seconds for idle speed.

The application allows for interaction with the canvas by moving it with the mouse (pulling with the hand tool) and zooming in and out with the mouse wheel. Further controls to move the timeline and to pause the animation are possible but have not been realized yet at the time of this writing.

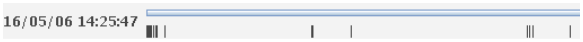


Figure 24: Enlarged timeline display of Figure 23

Figure 23 shows a typical situation of a few SNMP managers polling periodically a set of SNMP agents. The nodes are identified by labels indicating whether they act as a manager (labels starting with m) or an agent (labels starting with a). The timeline shown at the bottom of the window indicates when changes happen in the topology. Figure 24 show a zoom-in on the timeline of the configuration of Figure 23. The topology changes at the beginning of the trace are usually caused by adding nodes and edges until a stable situation has been established and as such usually do not indicate events of special interest. (An option to address this could be to suppress topology change events in the first $t_{startup} = max(t_{node}, t_{edge})$ seconds).

Our experience using this tool and, in particular, the adaptive replay mechanism has been very positive. In fact, the activity timeline itself gives valuable insight into the traces since changes to the topology-change dynamic graphs likely indicate events of some significance. A shortcoming of the current implementation is, however, that the removal of nodes and edges can lead to rather drastic changes of the graph layout. To deal with this, we are experimenting with strategies where nodes and edges first become invisible for a while so that a node / edge disappearing and reappearing shortly later does not cause instabilities in the graph layout.

5.2.3 Application to NetFlow Traces

The tools described in Sections 5.2.1 and 5.2.2 were originally developed to visualize SNMP traces. It turned out that it was straightforward to modify the translators so that they can parse NetFlow [53] traces and produce intermediate files for the visualization tools. Of course, flow traces collected at backbone routers need to be filtered and / or aggregated appropriately for the tools to produce meaningful node interaction visualizations.

For the `nam` conversion, some adaptations are required since flow traces only record the start and end of a flow, plus the number of bytes and packets exchanged. In order to enhance the visualization, we decided to choose the size and density of the visualized packets solely based on the duration of a flow. The source emits packets as long as the flow is present. The packets travel slowly in order to enhance the visualization. This also has the nice effect that very short flows will still be visible, so it is possible to go faster through an animation without losing a lot of details.

We are currently interested to understand flow patterns generated by personal hosts and specific applications. In particular, we like to answer the question to what extend flow patterns can be used to identify machines or even users. We assume that flow pattern generated by a certain host or user can serve as a fuzzy fingerprint and we are trying to explore techniques to identify hosts or users based on such fuzzy fingerprints. In order to evaluate whether this is feasible, we started an effort to collect flow records on end user devices such as notebooks. These traces are reasonably small in size and with some limited filtering one can achieve useful visualizations of interaction dynamics in order to identify characteristic patterns.

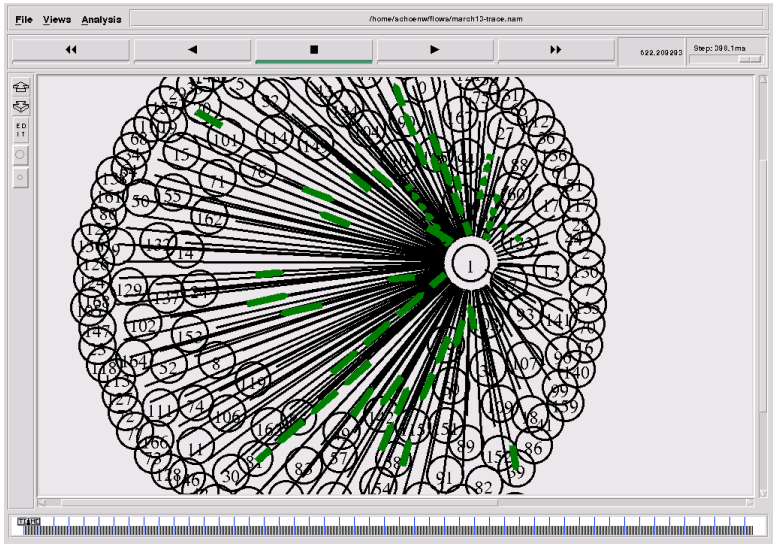


Figure 25: Visualization of NetFlow traces using nam

Figure 25 shows a screenshot of nam visualizing the traffic generated by a notebook during one day. The screenshot shows some Web browsing activity of the user of the notebook at a certain point of time. By moving through the timeline, it is possible to easily identify periods of high activity and sites visited frequently. To further improve the visualization results, it is useful to pre-process the NetFlow traces by aggregating IP addresses so that, for example, different IP addresses of Google servers appear as a single node.

5.2.4 Conclusions

Most existing visualizations of network traces focus on the static properties of the traces. Since our goal is to understand the dynamics recorded in network traces, we did some experiments to create visualizations of node interaction dynamics using readily available tools or libraries.

We found that with relatively little effort, it is possible to adapt tools or libraries to produce meaningful visualizations of interaction dynamics. However, effective integration of the tools / libraries remains a major problem. To be effective, a network analyst must be supported by multiple different visualization views of the same data and user interactions must be reflected on all views. To achieve this, much more development work is needed.

Our experience with nam is two-fold: On the one hand, it was very easy to get started since the nam intermediate file format is easy to generate and debug. However, the graph layout algorithm used by nam is not producing very satisfying results for larger graphs (e.g., ineffective use of the canvas space and many overlapping nodes) and the zooming capabilities help to to deal with this shortcoming only to some extend. Furthermore, we found that the speed of the animation does not scale well for larger trace files. As the nam tool is written in Tcl/Tk (with some parts in C/C++), it is not easy to extend it since the effort of learning how the nam tool has been implemented is relatively high.

Our experience with the JUNG graph drawing framework has been mixed as well. The graphs produced by the JUNG library generally look nice and using the Java library is

rather straightforward due to good documentation and many readily available examples. While the performance of the graph layout algorithms is generally good, we did experience performance difficulties when we tried to animate more complex graphs. This is mainly a Java limitation since drawing on a canvas is relatively costly and does not exploit the capabilities of modern graphics hardware.

While we started with SNMP traces with very specific properties (dominant regular polling traffic with a relative stable communication matrix), it turned out that some of the tools we created could be adapted easily to deal with other trace formats. The decoupling of visualization software from specific data sources through intermediate file formats has proven to be a big win here. Unfortunately, some other openly available visualization tools we looked at do not support such intermediate formats and integrating their visualization capabilities or adapting these tools to different trace formats requires much more involved programming efforts.

5.3 Visualization Tools

Visualization allows to better understand and easily explore the data. Beginners, which have no experience with visualization, can start to visualize their data using DAVIX, a live CD for data analysis and visualization. It brings the most important free tools for data processing and visualization to the desk. There is no hassle with installing an operating system or struggle to build the necessary tools to get started with visualization. This section is the try to classify and categorize all visualization tools available in DAVIX.

Network visualization tools can produce as an output different forms of graphical information. These forms can be graphs, charts, parallel coordinates charts, multidimensional cubes, and other forms. The following sections discuss tools classified according to these forms in more detail.

5.3.1 Graphs

Graph visualization is a way of representing structural information as diagrams of abstract graphs and networks. Automatic graph drawing has many important applications in software engineering, database and web design, networking, and in visual interfaces for many other domains.

Name of the tool	Support for visualization of any data/ local network/ whole network/ other specify	Homepage
AfterGlow	N/Y/Y/Y	`http://afterglow.sourceforge.net/`
Cytoscape	Y/N/N/N	`http://www.cytoscape.org/`
EtherApe	N/Y/N/N	`http://etherape.sourceforge.net/`
Graphviz	Y/N/N/N	`http://www.graphviz.org/`
GUESS	Y/N/N/N	`http://graphexploration.cond.org/documentation.html`
Large Graph Layout (LGL)	Y/N/N/N	`http://lgl.sourceforge.net/`
RT Graph 3D	Y/Y/N/N	`http://www.secdev.org/projects/rtgraph3d/`
Tulip	Y/N/N/N	`http://www3.labri.fr/perso/auber/projects/tulip/`
Walrus	Y/N/N/N	`http://www.caida.org/tools/visualization/walrus/`

Graphs tools overview:

AfterGlow is a collection of scripts facilitating the process of generating link graphs. AfterGlow 1.x is written in Perl and is meant to be used on the command line. There is no graphical user interface available. AfterGlow expects a CSV file as input and generates either an attributed graph language DOT file that can be processed by the Graphviz libraries, or it can generate output for consumption by the large graph library (LGL) or Walrus. AfterGlow 2.0 is written in Java and generates treemaps. It takes the same kind of input as AfterGlow 1.x.

CSV files can be created from logs of the following tools: argus(IP Audit tool), ipfw (Mac OS firewal), pf (*BSD firewal), sendmail (mail server), snort (network intrusion prevention and detection system), tcpdump (capturing packets tool)

Cytoscape is an open source bio-informatics software platform for visualizing molecular interaction networks and biological pathways and integrating these networks with annotations, gene expression profiles and other state data. Although Cytoscape was originally designed for biological research, it is now a general platform for complex network analysis and visualization.

EtherApe is a graphical network monitor for Unix modelled after etherman (Ethernet Man In The Middle Attack Tool). Featuring link layer, IP and TCP modes, it displays network activity graphically. Hosts and links change in size with traffic and it uses a colour coded protocol display. It supports Ethernet, FDDI, Token Ring, ISDN, PPP and SLIP devices. It can filter traffic to be shown, and can read traffic from a file as well as live from a network interface.

Graphviz is open source graph visualization software. It has several main graph layout programs. They take descriptions of graphs in a simple text language, and make diagrams in several useful formats such as images and SVG (Scalable Vector Graphics) for web pages, Postscript for inclusion in PDF or other documents; or display in an interactive graph browser; Graphviz also supports GXL, an XML dialect. Graphviz has many useful features for producing useful diagrams, such as options for colors, fonts, tabular node layouts, line styles, hyperlinks, and custom shapes.

GUESS is an exploratory data analysis and visualization tool for graphs and networks. The system contains a domain-specific embedded language called Gython (an extension of Python, or more specifically Jython), which supports the operators and syntactic sugar necessary for working on graph structures in an intuitive manner. An interactive interpreter binds the text that the user types in the interpreter to the objects being visualized for more useful integration. GUESS also offers a visualization front end that supports the export of static images and dynamic movies.

LGL is a compendium of applications for making the visualization of large networks and trees tractable. LGL was specifically motivated by the need to make the visualization and exploration of large biological networks more accessible. Essentially the network is a graph and LGL is responsible for presenting it to the user.

RTgraph3D lets users create dynamic graphs in 3D. It provides this ability as a D-Bus service. Programs can use it by registering to the bus, claiming its command interface, and issuing commands. Events like shift-clicking on a node are sent as signals on the bus. IPv6world is a program that runs a probe and interact with RTgraph3D to visualize IPv6 network. The probe uses scapy6 and the scapy6 teredo extension, which means that, to some extent, user can reach the IPv6 backbone on a box with only basic IPv4 access.

Tulip allows the visualization, the drawing and the edition of graphs having more than 1.000.000 elements. It allows the navigation through geometric operations as well as the extraction of subgraphs and the enhancement of the results obtained by filtering.

Walrus is a tool for interactively visualizing large directed graphs in three-dimensional space. It is technically possible to display graphs containing a million nodes or more, but visual clutter, occlusion, and other factors can diminish the effectiveness of Walrus as the

number of nodes, or the degree of their connectivity, increases. Thus, in practice, Walrus is best suited to visualizing moderately sized graphs that are nearly trees. A graph with a few hundred thousand nodes and only a slightly greater number of links is likely to be comfortable to work with. By employing a fisheye-like distortion, it provides a display that simultaneously shows local detail and the global context.

5.3.2 Charts

A chart is a visual representation of data such as tabular numeric data, functions or some kinds of qualitative structures.

Name of the tool	Support for visualization of any data/ local network/ whole network/ other specific	Homepage
ChartDirector	Y/N/N/N	http://www.advsofteng.com/
GNUplot	Y/N/N/N	http://www.gnuplot.info/
Mondrian	Y/N/N/N	http://rosuda.org/Mondrian/
MRTG	Y/Y/Y/Y	http://oss.oetiker.ch/mrtg/
Ploticus	Y/N/N/N	http://ploticus.sourceforge.net/doc/welcome.html
R Project	Y/N/N/N	http://www.r-project.org/
RRDtool	Y/N/N/N	http://oss.oetiker.ch/rrdtool/
Timesearcher 1	Y/N/N/N	http://www.cs.umd.edu/hcil/timesearcher/

Charts tools overview:

ChartDirector is commercial component for the Windows operating system and web applications to easily create a wide variety of charts. There are libraries available for ASP/COM/VB, Java/JSP, ColdFusion, PHP, Perl, Python, Ruby and C++.

Gnuplot is a portable command-line driven graphing utility for Linux, OS/2, MS Windows, OSX, VMS, and many other platforms. The source code is copyrighted but freely distributed (i.e., users do not have to pay for it). Gnuplot was originally created to allow scientists and students to visualize mathematical functions and data interactively, but has grown to support many non-interactive uses such as web scripting. It is also used as a plotting engine by third-party applications like Octave. Gnuplot has been supported and is under active development since 1986.

Gnuplot supports many types of plots in either 2D and 3D. It can draw using lines, points, boxes, contours, vector fields, surfaces, and various associated text. It also supports various specialized plot types.

Mondrian is a general purpose statistical data-visualization system. It features visualization techniques for data of almost any kind, and has its particular strength compared to other tools when working with categorical data, geographical data and large data.

All plots in Mondrian are fully linked, and offer various interactions and queries. Any case selected in a plot in Mondrian is highlighted in all other plots. Currently implemented plots

comprise Mosaic Plot, Scatter-plots and SPLOM, Maps, Barcharts, Histograms, Missing Value Plot, Parallel Coordinates/Boxplots and Boxplots y by x.

MRTG is monitoring SNMP enabled network devices and draws charts showing how much traffic has passed through each network interface. Routers are only the example. MRTG is being used to graph all sorts of network devices as well as everything else from weather data to vending machines (data acquisition not only via SNMP queries, but also via external scripts).

MRTG is written in Perl and works on Unix/Linux as well as Windows and even Netware systems. MRTG is free software licensed under the Gnu GPL. MRTG can (and it is recommended) use the RRD data format.

Ploticus is a free, GPL, non-interactive software package for producing plots, charts, and graphics from data. It was developed in a Unix/C environment and runs on various Unix, Linux, and Win32 systems. Ploticus is good for automated or just-in-time graph generation, handles date and time data nicely, and has basic statistical capabilities. It allows significant user control over colours, styles, options and details.

R Project is a language and environment for statistical computing and graphics. It is a GNU project, similar to the S language and environment, which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, etc.) and graphical techniques, and is highly extensible.

RRDtool is an open source industry standard, high performance data logging and graphing system for time series data. RRDtool was developed out of the MRTG package.

Timesearcher 1 is interactive tool for querying and exploration of time-series data, discovering features and trends. Queries are built using timeboxes: a powerful graphical, direct-manipulation metaphor for the specification of queries over time-series data sets.

5.3.3 Parallel Coordinates Charts

Parallel coordinate charts are a method of visualizing multivariate data. An n-dimensional space is represented as n parallel lines. A point in n-dimensional space is represented as a poly-line with vertexes on the parallel axes; the position of the vertex on the i-th axis corresponds to the i-th coordinate of the point.

Name of the tool	Support for visualization of any data/ local network/ whole network/ other specific	Homepage
GGobi	Y/N/N/N	`http://www.ggobi.org/`
Parvis	Y/N/N/N	`http://home.subnet.at/flo/mv/parvis/`
rumint	N/Y/Y/N	`http://www.rumint.org/`

Parallel Coordinates Charts tools overview:

GGobi is an open source visualization program for exploring high-dimensional data. It provides highly dynamic and interactive graphics such as tours, as well as familiar graphics such as the scatter-plots, bar-charts and parallel coordinates plots. Plots are interactive and linked with brushing and identification.

Parvis is a component compliant to the Java Swing and Java Beans for parallel coordinates (PC) visualisation of multidimensional data sets.

rumint is an open source network and security visualization tool written for the Windows operating system (under Linux it can be run using wine), which offers following functionality: Loading of pcap data sets and capturing live traffic, VCR/PVR interface to play back the traffic, visualization of packets in seven carefully designed windows. It is extremely flexible with a total of ≈20 different views and it currently handles up to 30,000 packets in a high speed RAM buffer.

5.3.4 Cube

A cube is a way of visualizing network traffic in a 3d visualization. The x, y, and z axes correspond to the source IP address, the port number, and the destination IP address, respectively. Such visualization seems to help humans in identifying port scans and the like.

Name of the tool	Support for visualization of any data/ local network/ whole network/ other specific	Homepage
InetVis	N/Y/Y/N	`http://www.cs.ru.ac.za/research/g02v2468/inetvis.html`
Shoki Packet Hustler	N/Y/Y/N	`http://shoki.sourceforge.net/`

Cube tools overview:

InetVis is a 3-D scatter-plot visualization for network traffic. In a way, it is more or less like a media player, but for network traffic. It is quite handy for observing scan activity and other anomalous traffic patterns.

Shoki Packet Hustler is a tool for the visualization of IP network data. In particular, it is intended to be useful in allowing an analyst to visually identify patterns in network traffic.

5.3.5 Other

We have already described many different tools offering a wide variety of functionality, but there are still some tools which could not be classified to any group mentioned before. These tools are also distributed with DAVIX and will be presented here.

Name of the tool	Support for visualization of any data/ local network/ whole network/ other specific	Homepage
glTail	N/N/N/Y	http://www.fudgie.org/
NVisionIP	N/Y/Y/N	http://security.ncsa.uiuc.edu/distribution/NVisionIPDownLoad.html
Processing	Y/N/N/N	http://processing.org/
Scapy	N/Y/Y/N	http://www.secdev.org/projects/scapy/
tnv	N/N/Y/N	http://tnv.sourceforge.net/
Treemap	Y/Y/Y/N	http://www.cs.umd.edu/hcil/treemap/
Wireshark	N/Y/Y/N	http://www.wireshark.org/

Other tools overview:

glTail visualizes in real-time data and statistics from any logfile on any server in an intuitive and entertaining way: falling balls from both sides, where one side reflects requests and the other responses. The color and size of the balls can indicate details of the event represented by the ball.

NVisionIP visualizes traffic flows to / from every machine on a large and complex computer network. It leverages the innate cognitive processing abilities of human operators, allowing them to see security events. In addition to an overall view of an entire network on one screen, NVisionIP also includes the ability to drill down multiple levels to view subnets of machines or view attributes on individual machines relevant to security such as connection and data transfer statistics per protocol or per port. NVision uses “galaxy” charts, where subnets are on x-axis and hosts are on the y-axis.

Processing is a simple programming environment that was created to make it easier to develop visually oriented applications with an emphasis on animation and providing users with instant feedback through interaction. It can be used to visualize graphs and also any other objects (charts, cubes) in 2-D or 3-D space.

Scapy is a powerful interactive packet manipulation program written in Python. It is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more. It can easily handle most classical tasks like scanning, tracerouting, probing, unit tests, attacks or network discovery (it can replace hping, 85% of nmap, arpspoof, arp-sk, arping, tcpdump, tethereal, p0f, etc.). It also performs very well at a lot of other specific tasks that most other tools cannot handle, like sending invalid frames, injecting your own 802.11 frames, or combining techniques (VLAN hopping + ARP cache poisoning, VOIP decoding on an WEP encrypted channel, ...).

Scapy uses external libraries to produce graphical output: PyX - to produce PDF/PS dumps; Gnuplot to plot some harvested values, for example, the IP ID patterns to know how many distinct IP stacks are used behind a load balancer; Graphviz to make a directed graph from all the routes taken by traceroute and to cluster them by AS; VPython - to have a 3D representation of a traceroute.

tnv is intended for network traffic analysis. The main visualization shows remote hosts along the left side and a reorderable matrix of local hosts on the right, with links drawn between them. The local host matrix shows aggregated packet activity as background color, and shows network packets as triangles, with the point representing the directionality of the packet. tnv can open saved libpcap (from tcpdump, windump, ethereal, etc.) formatted files or capture live packets on the wire, and export data in libpcap format or save the data to a MySQL database to enable examining trends over time.

Treemap is a space-constrained visualization of hierarchical structures. It is very effective in showing attributes of leaf nodes using size and color coding. Treemap enables users to compare nodes and sub-trees even at varying depth in the tree, and help them spot patterns and exceptions. The NetViz feature allows to have links between leaf nodes. There are no arrows to indicate incoming and outgoing links. Instead, the curvature of the link is used to determine that. The curve is nearer to the source node.

Wireshark is the world's foremost network protocol analyzer. It is the de-facto standard across many industries and educational institutions. Wireshark's main functionality covers capturing and analysis; the offered visualization is limited to grouping and colouring captured packets presented in textual form.

5.4 SURFmap: A Network Monitoring Tool based on the Google Maps API

5.4.1 Introduction

Computer networks are complex communication systems that enable intensive interactions among users and services. Such interactions result in network traffic that should be monitored to check the health of the underlying communication infrastructure. Network monitoring, in addition, provides essential data to other network management processes, such as network optimization, accounting, and security. As a result, network monitoring turns out to be a critical task in any serious network management solution.

Network monitoring tools help network managers (or other end users) to handle network information. They usually provide some kind of user interface that presents network information. Nowadays, many network monitoring tools are made available, each one of them having a particular purpose and highlighting different aspects of network information. Some network monitoring tools focus on the inspection of packets, for example, whereas others focus rather on the monitoring of flows. Network managers often choose a particular network monitoring tool for a particular purpose. It is desired although that the chosen monitoring tool presents network information in a clear and easy way.

However, many of the current network monitoring tools lack to provide network traffic information at a geographical dimension. For some network manager's tasks, (e.g., attack tracking and user statistics) this feature could be desirable. Moreover, if a network manager would like, for instance, to get an overview of the geographical locations of network traffic passing by his/her managed network, he/she could get a simple but complete overview when a map, containing the geographical locations, is displayed. In this paper, we propose a monitoring tool prototype that allows network managers to visualize network information through multiple levels of abstraction by using geographical capabilities provided by the Google Maps API[6]. The main contribution of this work is that it provides network traffic information in a geographical dimension.

Note that the following description of SURFmap has been also published in [54].

5.4.2 Related work

Most of the current network monitoring tools are developed for showing specific aspects of network traffic. The way in which network information should be presented can although be different for each purpose. Generally, most of the tools attend specific requirements on how and which network information is presented.

Wireshark[7] is a packet sniffer and protocol analyzer. It is able to capture and display all network traffic transiting at a particular network adapter. Furthermore, Wireshark recognizes the used protocols and is therefore able to put network information into its context. Tcpdump[8] is also a packet sniffer, but runs under a command line interface. The information provided by Tcpdump is basically the same as from Wireshark, but the latter provides

[6] http://code.google.com/apis/maps/
[7] http://www.wireshark.org
[8] ftp://ftp.ee.lbl.gov/

a graphical user interface, whereas Tcpdump provides a textual one. However, both tools lack to provide any geographical information about network traffic.

At the level of flow information, monitoring tools such as ntop[9] and NfSen[10] are available. The main purposes of these tools are traffic measurement (measuring the usage of a particular network and maintaining statistics of that), traffic monitoring and network security issue detection. While both of these tools can provide a graphical user interface, they do not provide network traffic in a geographical dimension.

In the past few years, Google Maps has shown to have more potential than simply providing satellite imagery that allows users to drive directions in a fashion way. Google Maps has been widely used in several other areas (see `http://googlemapsmania.blogspot.com/` for an overview) by means of the free usage of its API (Application Programming Interface). In the field of network monitoring, some research works have also been done. Van der Ham et al. [55] investigated the usage of the Google Maps API to visualize optical path finding in the GLIF infrastructure[11]. In another work, developed by Jamjoom et al. [56], the authors focus on the use of Google Maps' capabilities for the monitoring and management of network infrastructures.

The main drawback of the considered works is that they do not focus on the visualization of network traffic, even though some of them use the Google Maps API for their purposes. Since the Google Maps user interface has proven to be very intuitive by many different user groups, it could also be a very good starting-point as a new user interface for a network monitoring tool. Note that SURFmap is not meant to be a replacement for the current network monitoring tools, but to be an alternative.

5.4.3 Our approach

Since current network monitoring lack to provide geographical information about network traffic, we have developed a prototype tool that is intended to overcome this issue. Our tool, called SURFmap, provides a graphical user interface integrated with the Google Maps API. The latter has proven to be easy to use and provides map navigation functions in a fashion way. In addition, the zooming functions of the Google Maps API are implemented in SURFmap, in order to provide different levels of detail for the considered network information.

At its current development stage, SURFmap uses network information provided by SURFnet [57] from the Dutch research and education network, SURFnet6 [58]. This network information is collected from the SURFnet6 routers by using NetFlow [59] and correlated in a central storage database. SURFmap obtains the network information from this database and uses elements of the Google Maps API in order to visualize them.

In order to add a geographical dimension, SURFmap combines the network information obtained from SURFnet6 with the geographical information provided by IP2Location[12]. IP2Location provides several database sets to identify host's geographical locations (e.g.,

[9] `http://www.ntop.org`
[10] `http://nfsen.sourceforge.net`
[11] `http://www.glif.is`
[12] `http://www.ip2location.com`

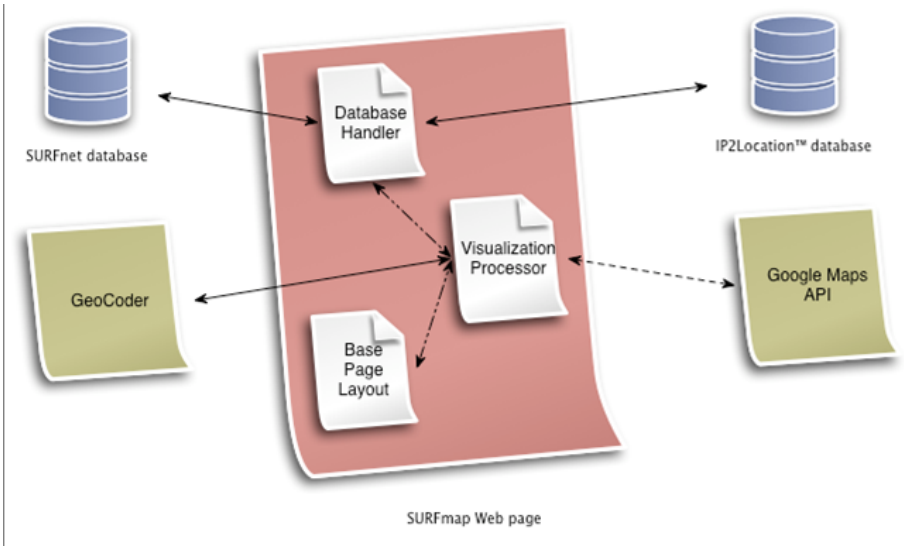


Figure 26: SURFmap architecture

country, city, latitude, longitude, and so on). SURFmap divides the geographical information into several zoom levels and uses these zoom levels to show network information with different levels of detail.

Moreover, SURFmap provides four zoom levels: country zoom level, region zoom level, city zoom level and host zoom level of information. SURFmap enables its users to zoom in and out, through these zoom levels depending on the amount of details a user wants to obtain regarding the considered network traffic.

SURFmap architecture

The system architecture of SURFmap (Figure 26) is separated into multiple parts. The main part is the SURFmap Web page which is the core of our tool, since it manages all other external parts. This Web page consists of three subparts: the Database Handler, the Visualization Processor and the Base Page Layout.

The Database Handler is responsible for the communication with both databases used in the development of this prototype. One of them is called the “SURFnet database”, which contains all actual network data provided by SURFnet. The other database is called “IP2Location database”, which contains geographical information about IP addresses and IP ranges. The Visualization Processor receives the network and geographical information from the Database Handler. After that, the GeoCoder is used to retrieve the coordinates (latitude and longitude) of the geographical information. SURFmap uses the GeoCoder provided by the Google Maps API. Another task of the Visualization Processor element is the creation of a map with its elements, using the Google Maps API; after formatting the network information, the Google Maps API’s elements such as markers and lines are created using the geocoded information.

At last, the Visualization Processor sends the created map to the Base Page Layout, which puts it into a standard Web page with some extra elements, such as a help function and some navigation support.

The next subsections present detailed descriptions of the system parts presented above, as well as some application examples.

Obtaining information from the databases

In order for SURFmap to show network traffic information, the Database Handler first needs to obtain the required information from the SURFnet and IP2Location databases. The SURFnet database contains the network information to be used by SURFmap, stored per flow and having the following fields: start_time, end_time, duration, ipv4_src, port_src, ipv4_dst, port_dst, protocol, packets and octets.

The first step performed by the Database Handler is obtaining network information from the SURFnet database by using SQL commands. The network information is then returned to the Database Handler, which takes all IPv4 addresses per flow (both source and destination addresses) and uses them to obtain the geographical information from the IP2Location database. Geographical information of both the sending and receiving host (in a flow) are obtained in this step. The information is stored in IPv4 address ranges and has the following fields: ip_from, ip_to, country, region, city, latitude and longitude. As the second step, the Database Handler obtains the geographical information about the network information collected in the first step. The IPv4 source and destination addresses obtained from the SURFnet database are used as input for the SQL query for the IP2Location database.

As an example, let us imagine one FTP communication from the host 1.2.3.4:21 (numerical representation: 16.909.060) to the host 5.6.7.8:32657 (numerical representation: 184.551.176) which lasted 1 hour (3.600.000 ms) and transferred 3MB (3.145.728 octets) of data within 1.000 packets. The Database Handler would first query the SURFnet database and obtain a row like this: 1; 3600000; 16909060; 21; 184551176; 32657; 6; 1000; 3145728. The source and destination IP addresses would then be used by the Database Handler to query the IP2Location database. The result could be, for example: “1; 16777216; 16909568; Brazil; Rio Grande do Sul; Porto Alegre; 30° 2’S; 51° 13’W” and “1; 83886080; 84281344; The Netherlands; Noord-Holland; Amsterdam; 52° 23’N; 4 55’E”. It is worth of saying that the IPv4 addresses are coded in the same way in both databases.

Processing information

The next step to be taken by SURFmap is the processing of information received from the Database Handler. Since the IP2Location database only provides the latitude and longitude coordinates of the last known zoom level, which is a limitation of the IP2Location database, the coordinates for the other levels have to be retrieved separately, using the GeoCoder.

In order to display network information consistently, the information obtained from both the GeoCoder and the two databases have to be converted into some proper display format. For example, the IP2Location database returns “NEW YORK”, which is converted into “New York”. Similarly, the SURFnet database offers flow octets, which requires SURFmap to convert it into Megabytes (depending on the zoom level).

In general, the GeoCoder assigns geographic identifiers (i.e., geographical coordinates) to an arbitrary kind of data (e.g., country names). In order for the GeoCoder to process certain fields, such as the city name, SURFmap has to make an asynchronous call to the GeoCoder by providing that field name as an argument. The GeoCoder then returns the latitude and longitude coordinates for the supplied argument. At the end, the Google

Maps API can use the provided latitude and longitude coordinates to place elements (e.g., markers) on the map.

As an example, let us imagine a host located at the University of Twente, which is located in Enschede, The Netherlands. The country name (provided by the IP2Location database) would then be “THE NETHERLANDS”, the region name “OVERIJSSEL” and the city name “ENSCHEDE”. To geocode this location, we have to make two calls to the GeoCoder: 1) for the country name and 2) for the region name (the city’s coordinates are already available, because that is the last known zoom level). The GeoCoder will then return two results, where the first contains the coordinates “52.132633” and “6.89114”, which are the coordinates for “THE NETHERLANDS” and the second contains the coordinates “52.436132” and “6.42529”, which are the coordinates for “OVERIJSSEL”.

Visualizing network information

After all the information is obtained from the databases by the Database Handler, geocoded by the Geocoder and converted by the Visualization Processor, the Visualization Processor displays the processed network traffic information. Following the SURFmap workflow, the processed information available at this stage is the following:

- network information per flow;
- geographical information (country, region and city names) of end points per flow;
- coordinates (latitude and longitude) of the geographical information.

Since the Google Maps API has two main display components, markers and lines, we decided to make a clear distinction between them. Markers represent hosts and show information about them, such as their IPv4 addresses, the country, region and the city they belong to. On the other hand, lines represent a flow between two hosts (so between markers) and show information about that flow, such as the used ports at the flow’s end points, the exchanged amount of packets, octets and throughput. The shown information is placed in information windows, which is an element provided by the Google Maps API. These information windows can be opened by clicking either a marker or a line.

Note that information about flows with the same end points are bundled together into one entry. As an example, two flows from “The Netherlands” to “United States” are written in the information window only once, but with the flow field counting as “2”. It is also worth of saying that SURFmap considers flows as unidirectional entries.

SURFmap zoom levels

As a consequence of the geographical dimension presented in SURFmap, our tool provides four zoom levels. The idea is that an application user can zoom in if he/she wants to get more information about some flow or host. The four zoom levels provided by SURFmap are:

- Country zoom level;
- Region zoom level;
- City zoom level;

Figure 27: Initial page of SURFmap

- Host zoom level.

These levels depend on the information provided by the IP2Location database. As explained before, this database does not contain information about all fields for all IPv4 address ranges. In that case, the geographical information of the last known zoom level is used. From now on, we assume the country level to be the highest zoom level and the host level to be the lowest one.

The country level represents all countries which provide some network activity in the user-specified amount of flows. The same applies to regions, cities and hosts, so the region level represents all regions which provide network activity, the city level represents the cities and the host level the hosts in the user-specified amount of flows.

A user is able to zoom in or out using two different steps: “zoom” is the ability to zoom in or out using the zoom steps provided by the Google Maps API. These steps are smaller than the steps provided by the “quick zoom” feature of SURFmap. These “quick zoom” steps are the steps mentioned above. Each of these steps consists of three zoom steps provided by the Google Maps API.

5.4.4 Application prototype

Figure 27 shows a screenshot of the initial page of SURFmap, which consists of the following elements:

- a world map, provided by the Google Maps API. This is the main screen of SURFmap, that shows the visualization of the network information at the selected level of detail;
- a table, showing all available zoom levels and the current zoom level (marked red);
- a help button, which provides guidelines on how to use SURFmap;

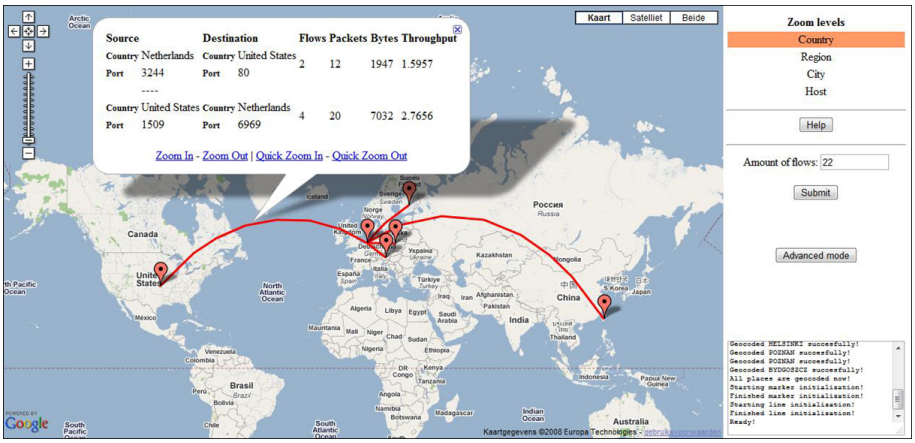


Figure 28: Country zoom level

- an advanced mode button, which enables a SURFmap user to edit more advanced SQL queries.

In order to start visualizing network information, the user selects the amount of flows to be displayed. The chosen amount of flows will then be selected from the SURFnet database following the order in which the flows were stored. Once the flows are selected, SURFmap retrieves the geographical information of these flows from the IP2Location database. At this stage, the user can start inspecting the flows information at their highest zoom level (i.e., country zoom level) or change to a different one. If the user decides to change the zoom level, the table to the right of the map will indicate the current zoom level. It is also possible to click at one of the zoom levels in the table to go to the selected zoom level directly. In short, the user is able to zoom in or out in three different ways:

- using the zoom in and out links in the information windows;
- using the table to the right of the map;
- using the zoom “slider” at the left top of the map, provided by the Google Maps API.

In the next subsections, we present three examples on the usage of SURFmap.

Zoom level grouping example

As mentioned before, SURFmap provides a zoom level grouping feature. This means that all active entities at a certain zoom level are fit together into one marker or line.

In Figure 28, SURFmap is showing that there is network activity between the United States and The Netherlands. When clicking the red line, an information window pops up showing that there is communication from The Netherlands (NL) to the United States (US) and vice-versa. The window shows that there are two flows from The Netherlands to the United States, in which 12 packets with a total amount of 1.947 bytes are transferred. On the other way around, there are four flows from the United States to The Netherlands in which 20 packets with a total amount of 7.032 bytes were transferred. Even though the first two flows (NL $\rightarrow$ US) are in reality generated by different hosts, they are grouped into the same communication. Since the screenshot is taken at the country zoom level,

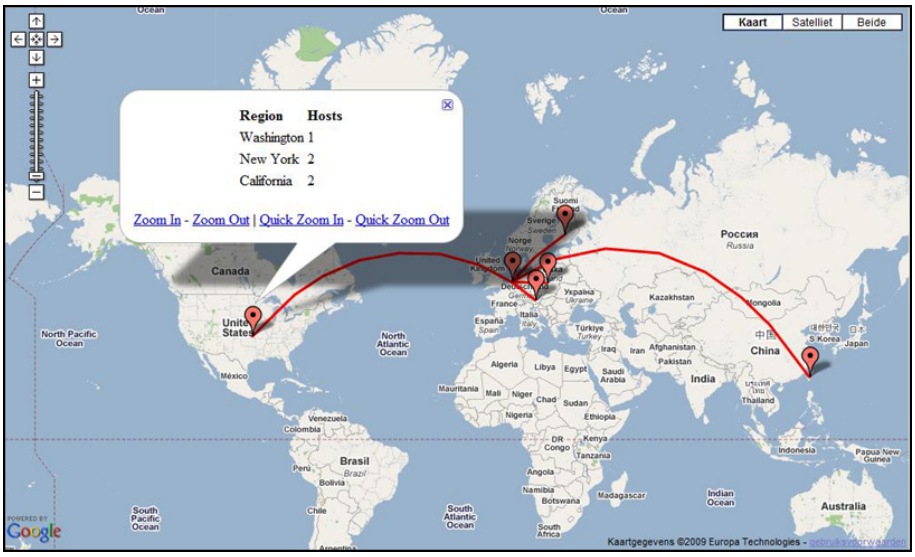


Figure 29: Region information at country zoom level

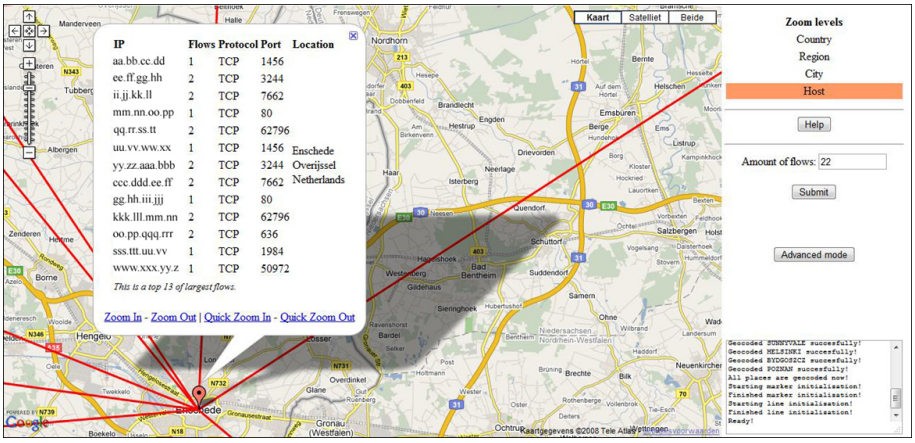


Figure 30: Host zoom level

zoom level grouping is based on the countries, in which the flow end points are located. The source countries of the first two flows are the same, just as the destination countries, which causes those two flows to be grouped together. The same counts for the last four flows from the United States to The Netherlands.

If the SURFmap user would like to know where in the United States the communication is coming from, he/she should click the marker representing the United States (Figure 29). Since SURFmap is at the country zoom level, the informative balloon shows information of one level lower, that is, the region zoom level. Combining therefore the information provided by both screenshots, the user knows that the regions Washington, New York and California are communicating with The Netherlands, either as a source or as a destination.

Host zoom level example

Figure 30 and 31 show the information provided by SURFmap at the host zoom level. Normally, the marker placed in Enschede shows all active hosts located in that city. However, since there are more active hosts than do fit into the information window, only a subset of them is shown here. The information provided by the information window are the IPv4

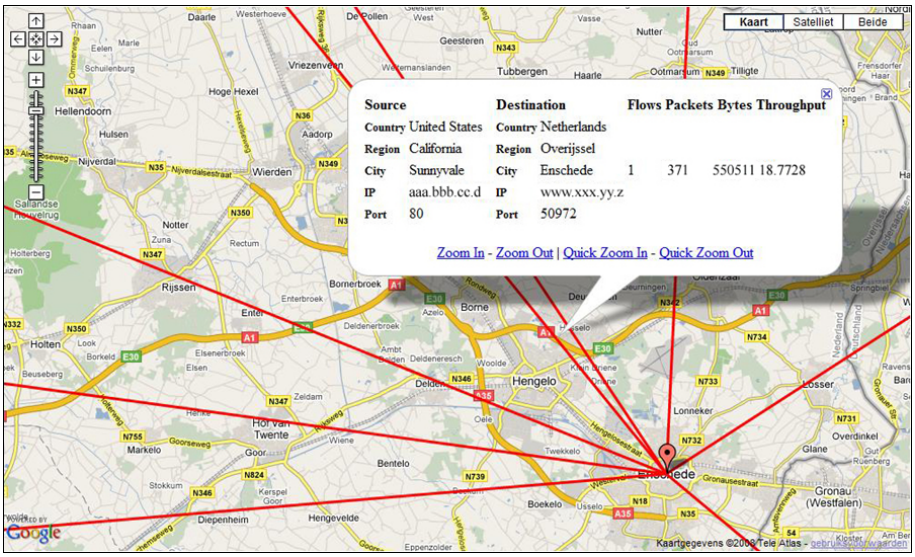


Figure 31: Region information at country zoom level

addresses of the hosts in Enschede, the amount of flows to or from each host, the flow protocols, the ports at which the host communicates and the geographical location of the hosts. Since they are all in the same city (and thus in the same country and region) this geographical information is the same for all hosts in this information window.

The line's information window shows that there is one flow between the aaa.bbb.cc.d[13] and www.xxx.yy.z hosts. It also shows that the host aaa.bbb.cc.d, which is located in Sunnyvale, California, United States, exchanged 371 packets containing 550.511 bytes of data with the host www.xxx.yy.z, which is located in Enschede, Overijssel, The Netherlands. The throughput of this flow was 18,7728 KBps. Besides that, we can conclude that this flow belongs to HTTP traffic, since the source port of the flow is 80.

The next subsection presents a concrete scenario in which SURFmap appears to be very useful for network monitoring.

Real-case example

In order to show the usability of SURFmap, we presented our prototype at SURFnet. SURFnet is a subsidiary of the SURF organisation, in which Dutch universities, universities for applied sciences and research centres collaborate nationally and internationally on innovative ICT (Information and Communication Technology) facilities. The presentation consisted of several cases, in which SURFmap appears to be very useful. One of them was a DNS attack on a University of Twente DNS server.

Figure 32 shows that several places were involved in the DNS attack. One of the attacker (or reflector) hosts was located in the United States. Figure 33 shows that this host (having the eee.fff.gg.h IPv4 address) was located in Los Angeles, California. Besides that, SURFmap shows that there were 63 flows between this attacker host and the Dutch DNS server, in which 2.772.949 packets containing 119.236.807 bytes were exchanged. This means that the packets had an average size of 43 bytes.

[13] Due to a non-disclosure agreement signed with SURFnet, real IP addresses cannot be shown. The real IP addresses were therefore replaced by fictional IPv4 addresses (e.g., aaa.bbb.cc.d).

Figure 32: DNS attack (country level)

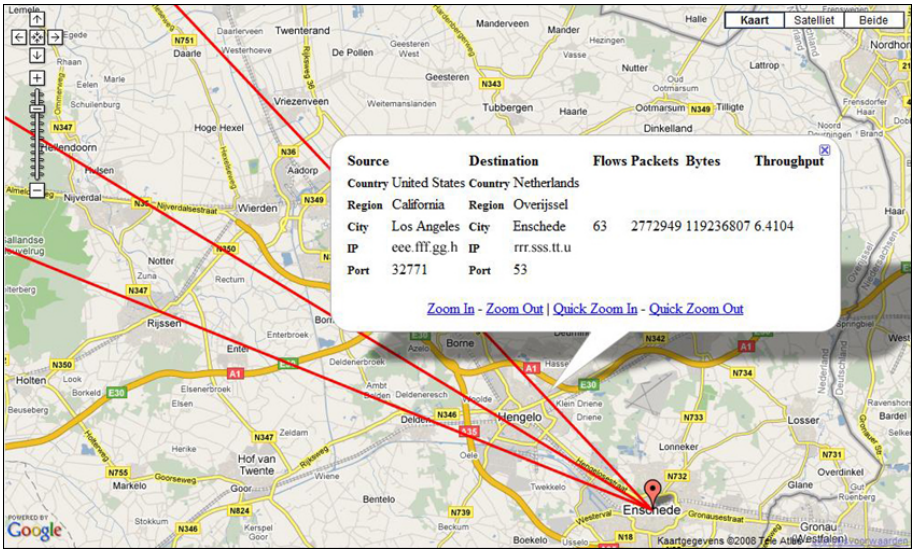


Figure 33: DNS attack (host level)

After presenting our prototype tool, SURFnet gave us the following feedback to improve SURFmap: 1) let the line colors depend on the data volume of particular flows, and 2) integrate NfSen to let SURFmap show semi-realtime network traffic information every 5 minutes.

Conclusions

In this paper we presented a network monitoring tool prototype that shows network information using the Google Maps API. Our motivation for developing this tool was the fact that current monitoring tools lack to provide geographical information about network traffic. Since the Google Maps user interface seems to be popular in many different user groups, the development of a network monitoring tool using the Google Maps API could make network information easier to understand. Our monitoring tool prototype, SURFmap, has shown to be very intuitive when displaying network traffic information.

Following the recent trend of adding a geographical dimension to ordinary information (e.g., geolocation for vineyards), SURFmap adds a geographical dimension to network data. By adding this new dimension, SURFmap provides a totally different view on network information, compared to the information provided by current network monitoring tools. Moreover, besides using Google Maps API's elements such as markers to represent hosts, lines to represent flows and information windows to show information about either hosts or flows, SURFmap provides four different zoom levels (country, region, city, and host zoom levels) to more easily show network information about hosts and flows. By providing those zoom levels, SURFmap users can intuitively obtain more or less information, which makes it an important feature of our tool.

As future work, we intend to improve the way lines and markers are shown by SURFmap. Especially the animation of links, e.g. different line colors and thickness to make a clearer distinction between the various zoom levels and to indicate the network load, will be considered. In addition, we are also planning to integrate NfSen into SURFmap, because NfSen is a very popular network monitoring tool, which provides semi-realtime network traffic information every 5 minutes and offers the ability to filter this information quite fast.

Last but not least, we believe that the main contribution of this work is that SURFmap gives network managers a better, visualized network overview by providing network traffic information in a geographical dimension.

6 Conclusion

The sixth "Virtual Laboratory Integration Report" documents the achievements of the three projects sponsored by WP2 during the last twelve-month period covering January 2009 until December 2009. All three projects achieved their goals defined in January 2009 and the open call process again proofed to be a good instrument for assigning funding in a timely, flexible and effective manner.

The major achievements of the three funded projects can be summarized as follows:

- The "EmanicsLab 3.0" has proved to be a very reliable and worthwhile virtual laboratory infrastructure. The establishment of an EmanicsLab charter and an EmanicsLab Steering Committee shows the strong interest of the involved partners to continue the shared infrastructure past the end of the EMANICS project. Through the planned federation with PlanetLab Europe, additional synergies on the European level can be achieved.
- The "Network Trace Collection and Labeling" activity led to an improved infrastructure, based on EmanicsLab, for sharing network traces. In addition, the first publically available labeled flow trace was published. Finally the consistency of packet traces has been investigated and it was shown that traces often contain anomalies.
- The "Trace Visualization and Anonymization Tools" project analyzed and developed trace data visualization and anonymization / deanonymization tools.

The "Virtual Laboratory and Common Testbeds" work package achieved its goals with two face-to-face meetings (January 2009 and July 2009). In addition, there were a number of informal exchanges that took place next to workshops or conferences the work package members attended for other reasons. The good working relationships established during the past EMANICS years have made communication and coordination in this work package very efficient.

7 Acknowledgement

This deliverable was made possible due to the large and open help of the WP2 Partners of the EMANICS NoE. Many thanks to all of them.

References

[1] Ganglia: EmanicsLab Node Usage Monitor. Available at `http://www.emanicslab.org/ganglia/`; Last accessed December 2009.

[2] Anna Sperotto, Ramin Sadre, Frank Vliet van, and Aiko Pras. A labeled data set for flow-based intrusion detection. In *IP Operations and Management (IPOM 2009)*, number 5843 in Lecture Notes in Computer Science, pages 39–50. Springer, October 2009.

[3] CERT Coordination Center. http://www.cert.org/certcc.html, Jan. 2009.

[4] P. Mell, V. Hu, R. Lippmann, J. Haines, and M. Zissman. An overview of issues in testing intrusion detection systems. Technical Report NIST IR 7007, National Insititute of Standards and Technology, June 2003.

[5] R.P. Lippmann, D.J. Fried, I. Graf, J.W. Haines, K.R. Kendall, D. McClung, D. Weber, S.E.Webster, D. Wyschogrod, R.K. Cunningham, and M.A. Zissman. Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In *Proc. of the DARPA Information Survivability Conf. and Exposition (DISCEX '00)*, 2000.

[6] R.P. Lippmann, J.W. Haines, D.J. Fried, J. Korba, and K. Das. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks*, 34, 2000.

[7] J.W. Haines, R.P. Lippmann, D.J. Fried, M.A. Zissman, E. Tran, and S.B. Boswell. 1999 DARPA Intrusion Detection Evaluation: Design and Procedures. Technical Report TR 1062, MIT Lincoln Laboratory, February 2001.

[8] J. Quittek, T. Zseby, B. Claise, and S. Zander. Requirements for IP Flow Information Export (IPFIX). RFC 3917 (Informational).

[9] C. Diot A. Lakhina, M. Crovella. Characterization of network-wide anomalies in traffic flows. In *Proc. of 4th ACM SIGCOMM Conf. on Internet measurement (IMC '04)*, 2004.

[10] A. Sperotto, R. Sadre, and A. Pras. Anomaly characterization in flow-based traffic time series. In *Proc. of the 8th IEEE International Workshop on IP Operations and Management, IPOM 2008, Samos, Greece*, Lecture Notes in Computer Science, September 2008.

[11] W.T. Strayer, D. Lapsely, R. Walsh, and C. Livadas. *Botnet Detection Based on Network Behavior*, volume 36 of *Advances in Information Security*, pages 1–24. Springer, 2008.

[12] H. Ringberg, A. Soule, and J. Rexford. Webclass: adding rigor to manual labeling of traffic anomalies. *SIGCOMM Computer Communication Review*, 38(1), 2008.

[13] H. Ringberg, M. Roughan, and J. Rexford. The need for simulation in evaluating anomaly detectors. *SIGCOMM Computer Communication Review*, 38(1), 2008.

[14] J. Sommers, V. Yegneswaran, and P. Barford. A framework for malicious workload generation. In *Proc. of the 4th ACM SIGCOMM Conf. on Internet measurement (IMC '04)*, 2004.

[15] D. Brauckhoff, A. Wagner, and M. Mays. Flame: a flow-level anomaly modeling engine. In *Proc. of the Conf. on Cyber security experimentation and test (CSET'08)*, 2008.

[16] F. Pouget and M. Dacier. Honeypot-based forensics. In *Asia Pacific Information technology Security Conference (AusCERT '04)*, May 2004.

[17] Citrix XenServer 5. http://www.citrix.com/, April 2009.

[18] OpenSSH. http://www.openssh.com/.

[19] `proftp`. http://www.proftpd.org/.

[20] Softflowd. http://www.mindrot.org/projects/softflowd/, April 2009.

[21] D. Moore, C. Shannon, D.J. Brown, G.M. Voelker, and S. Savage. Inferring internet denial-of-service activity. *ACM Trans. Comput. Syst.*, 24(2), 2006.

[22] R. Pang, V.Yegneswaran, P. Barford, V.Paxson, and L. Peterson. Characteristics of internet background radiation. In *Proc. of the 4th ACM SIGCOMM Conf. on Internet measurement (IMC '04)*, 2004.

[23] E. Lastdrager and A. Pras. Consistency of network traffic repositories: An overview. In *Proceedings of the Third International Conference on Autonomous Infrastructure, Management and Security (AIMS 2009), Enschede, The Netherlands*, volume 5637 of *Lecture Notes in Computer Science*, pages 173–178. Springer Verlag, July 2009.

[24] M. Timmer. How to identify the speed limiting factor of a tcp flow. `http://essay.utwente.nl/59138/`, September 2005.

[25] G. Slomp. Consistency of repositories. `http://referaat.cs.utwente.nl/new/paper.php?paperID=377`. Presented at 8th TSConIT.

[26] R. van de Meent and A. Pras. Simpleweb/university of twente – traffic measurement data repository. `http://traces.simpleweb.org`.

[27] Kenjiro Cho, Koushirou Mitsuya, and Akira Kato. Traffic data repository at the wide project. In *ATEC '00: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 51–51, Berkeley, CA, USA, 2000. USENIX Association.

[28] E.E.H. Lastdrager. Prototype and results. `http://www.vf.utwente.nl/~lastdragereeh/referaat`.

[29] A. Hanemann, J. Boote, E. Boyd, J. Durand, L. Kudarimoti, R. Lapacz, M. Swany, S. Trocha, and J. Zurawski. Perfsonar: A service oriented architecture for multi-domain network monitoring. In *Service-Oriented Computing - ICSOC 2005*, pages 241–254, Amsterdam, Netherland, 2005. Springer Verlag.

[30] Gant homepage including information about gn2 and gn3 projects. Available at `http://www.geant.net/`.

[31] Flow subscription measurement point homepage. Available at `https://wiki.man.poznan.pl/perfsonar-mdm/index.php/Flow_Subscription_MP`.

[32] Zebedee: Secure ip tunnel homepage. Available at `http://www.winton.org.uk/zebedee/`.

[33] Flow selection and aggregation measurement archive homepage. Available at `https://wiki.man.poznan.pl/perfsonar-mdm/index.php/Flow_Selection_and_Aggregation_MA`.

[34] The nfdump tools to collect and process netflow data homepage. Available at `http://nfdump.sourceforge.net/`.

[35] perfsonarui analysis tool homepage. Available at `http://iris.acad.bg/perfsonar/perfsonar.jnlp`.

[36] The Cooperative Association for Internet Data Analysis, Anonymization Tools Taxonomy. Available at `http://www.caida.org/tools/taxonomy/anonymization.xml`.

[37] Anonymization Application Programming Interface (AAPI). Available at `http://www.ics.forth.gr/dcs/Activities/Projects/anontool.html`.

[38] Tcpurify. Available at `http://irg.cs.ohiou.edu/~eblanton/tcpurify/`.

[39] Tcpdpriv. Available at `http://ita.ee.lbl.gov/html/contrib/tcpdpriv.html`.

[40] M. Harvan and J. Schönwälder. Prefix- and Lexicographical-order-preserving IP Address Anonymization. In *10th IEEE/IFIP Network Operations and Management Symposium*, pages 519–526, April 2006.

[41] Canine. Available at `http://security.ncsa.uiuc.edu/distribution/CanineDownLoad.html`.

[42] Flaim. Available at `http://flaim.ncsa.illinois.edu/`.

[43] Tcpmkpub. Available at `http://www.icir.org/enterprise-tracing/tcpmkpub.html`.

[44] M. Foukarakis, D. Antoniades, S. Antonatos, and E. P. Markatos. On the Anonymization and Deanonymization of NetFlow Traffic. In *Proc. of the FloCon 2008*, 2008.

[45] J. Schönwälder. Simple Network Management Protocol (SNMP): Traffic Measurements and Trace Exchange Formats. RFC 5345, Jacobs University Bremen, October 2008.

[46] J.G. van den Broek, J. Schönwälder, A. Pras, and M. Harvan. SNMP Trace Analysis Definitions. In *Proc. of the 2nd International Conference on Autonomous Infrastructure, Management and Security (AIMS 2008)*, number 5127 in LNCS. Springer, July 2008.

[47] J. Schönwälder, A. Pras, M. Harvan, J. Schippers, and R. van de Meent. SNMP Traffic Analysis: Approaches, Tools, and First Results. In *Proc. 10th IFIP/IEEE International Symposium on Integrated Network Management (IM 2007)*, pages 323–332, May 2007.

[48] P. Dobrev, S. Stancu-Mara, and J. Schönwälder. Visualization of Node Interaction Dynamics in Network Traces. In *Proc. 3rd International Conference on Autonomous Infrastructure, Management and Security (AIMS '09)*, pages 147–160. Springer-Verlag, 2009.

[49] G. van Rossum and F. Drake. *The Python Language Reference Manual*. Network Theory Limited, 2003.

[50] J. K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, April 1994.

[51] J. O'Madadhain, D. Fisher, P. Smyth, S. White, and Y.-B. Boey. Analysis and Visualization of Network Data using JUNG. journal paper under preparation.

[52] M. Campione and K. Walrath. *The Java Tutorial: Object-Oriented Programming for the Internet*. Addison Wesley, 2 edition, 1998.

[53] Cisco Systems. NetFlow Services Solution Guide, January 2007.

[54] R. J. Hofstede and T. Fioreze. Surfmap: A network monitoring tool based on the google maps api. In *2009 IFIP/IEEE International Symposium on Integrated Network Management (IM 2009), Long Island, New York, USA*, pages 676–690. IEEE Computer Society Press, June 2009.

[55] J. van der Ham, F. Dijkstra, P. Grosso, R. van der Pol, A. Toonk, and C. de Laat. A distributed topology information system for optical networks based on the semantic web. *Optical Switching and Networking*, 5(2-3):85–93, 2008.

[56] H. Jamjoom, N. Anerousis, R. Jennings, and D. Sana. Service assurance process re-engineering using location-aware infrastructure intelligence. In *10th IFIP/IEEE International Symposium on Integrated Network Management, 2007 (IM '07)*, pages 439–448, 2007.

[57] SURFnet. Pioneering network. `http://www.surfnet.nl/en/Pages/default.aspx`. Accessed on January 12, 2009.

[58] SURFnet. Map of the surfnet network. `http://www.surfnet.nl/en/netwerk/national/Pages/map.aspx`. Accessed on January 12, 2009.

[59] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954, Cisco Systems, October 2004.